



Instalação, configuração e administração.

Manual do utilizador

versão 1.0 Beta

Julho de 2020.

MiniBlocklyChain é uma marca registada da OpenQbit Inc., sob uma licença de uso livre e comercial. Termos e condições de utilização em: www.OpenQbit.com

Conteúdo

1.	Introdução.....	3
2.	O que é uma rede pública ou privada no esquema da cadeia de bloqueio?.....	4
3.	O que é a programação Blockly?.....	4
4.	O que é o Termux?	5
5.	O que é a Mini BlocklyChain?.....	5
6.	Arquitectura de processo em Mini BlocklyChain	10
7.	Diagrama de funcionalidade BlocklyChain (Mini BlocklyChain).....	13
8.	O que é o Mini BlocklyCode?	14
9.	Instalação de rede de comunicações Mini BlocklyChain	15
10.	Sincronização em nós de sistema (telemóvel) minutos e segundos.....	35
11.	Configuração de armazenamento dentro da Termux.....	42
12.	Instalação de rede "Tor" e instalação de "Syncthing".....	43
13.	Instalação da base de dados "Redis" e do servidor SSH (Secure Shell).	44
14.	Configuração do servidor SSH no telemóvel (smartphone).....	44
15.	Configuração de rede "Tor" com serviço SSH (Secure Shell).	51
16.	Configuração do sistema Peer to Peer com sincronização manual.	54
17.	Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).	65
18.	O que é a Prova de Quantum (PQu)?.....	66
19.	Definição e utilização de blocos em Mini BlocklyChain.	69
20.	Utilização de blocos para base de dados SQLite (versão MiniSQLite)	95
21.	Definição e utilização de blocos de segurança.....	99
22.	Definição de parâmetros de segurança em Mini BlocklyChain.....	109
23.	Anexo "Criação de bases de dados KeyStore & PublicKeys".....	113
24.	Anexo "Comandos RESTful SQLite GET/POST".....	125
25.	Anexo "Conector SQLite-Redis de código Java".....	131
26.	Anexo "Mini BlocklyChain para programadores".....	134
27.	Anexo "BlocklyCode Smart Contracts".....	146
28.	Anexo "OpenQbit Quantum Computing".....	152
29.	Anexo "Blocos alargados para base de dados SQLite".....	156
30.	Anexo "Exemplo de criação do sistema Mini BlocklyChain".....	156
31.	Anexo "Integração com ambientes Ethereum & Bitcoin".....	180
32.	Licenciamento e utilização de software.....	199

1. Introdução.

A cadeia de bloqueio é geralmente associada à moeda Bitcoin e outras moedas criptográficas, mas estas são apenas a ponta do iceberg uma vez que não é apenas utilizada para dinheiro digital, mas pode ser utilizada para qualquer informação que possa ter um valor para os utilizadores e/ou empresas. Esta tecnologia, que tem as suas origens em 1991, quando Stuart Haber e W. Scott Stornetta descreveram o primeiro trabalho sobre uma cadeia de blocos criptografados, só foi notada em 2008, quando se tornou popular com a chegada do bitcoin. Mas actualmente a sua utilização está a ser exigida noutras aplicações comerciais e prevê-se que cresça no futuro médio em vários mercados, tais como instituições financeiras ou a Internet das Coisas da Internet, entre outros sectores.

A cadeia de bloqueio, mais conhecida pelo termo cadeia de bloqueio, é um registo único, acordado e distribuído por vários nós (dispositivos electrónicos tais como PCs, smartphones, tablets, etc.) numa rede. No caso de moedas criptográficas, podemos pensar nisto como o livro de contabilidade onde cada uma das transacções é registada.

O seu funcionamento pode ser complexo de compreender se entrarmos nos detalhes internos da sua implementação, mas a ideia básica é simples de seguir.

É armazenado em cada bloco:

- 1.- uma série de registos ou transacções válidos,
- 2.- informações relativas a esse bloco,
- 3.- a sua ligação com o bloco anterior e o bloco seguinte através do hash de cada bloco –un código único que seria como a impressão digital do bloco.

Portanto, **cada bloco** tem um **lugar específico e inamovível dentro da cadeia**, uma vez que cada bloco contém informação do hash do bloco anterior. Toda a cadeia é armazenada em cada nó da rede que compõe a cadeia de bloqueio, pelo que **uma cópia exacta da cadeia é armazenada em todos os participantes da rede**.

À medida que novos registos são criados, são primeiro verificados e validados pelos nós da rede e depois adicionados a um novo bloco que está ligado à cadeia.

Agora imagine que esta rede de dispositivos que comunicam entre si, tem a capacidade de interagir sem a intervenção de uma pessoa, ou seja, a grande vantagem da cadeia de bloqueio é que pode tomar decisões autónomas, o que beneficia em tempos de resposta de serviço aos utilizadores, disponível 24 horas por dia, minimiza os custos nas empresas e, em primeiro lugar, tem um nível de segurança já testado, por esta e outras razões se tornou tão popular na utilização em diferentes sectores públicos e privados.

2. O que é uma rede pública ou privada no esquema da cadeia de bloqueio?

Rede pública. - É uma rede de computadores ou dispositivos móveis que comunicam entre si e mantêm o anonimato, não se sabe quem interage nesta rede de uma forma formal nas suas transacções, neste tipo de rede qualquer pessoa ou empresa pode interagir e ligar-se a qualquer momento porque não precisa de permissões para se ligar, um exemplo é a cadeia de bloqueio da Bitcoin, qualquer pessoa pode entrar para comprar ou vender. Normalmente este tipo de redes são orientadas ou dirigidas para a compra e venda de activos monetários digitais ou o seu sinónimo "criptomónias", exemplos: DogCoin, Ethereum, LiteCoin, BitCoin, Waves, etc.

Rede privada. - É uma rede de computadores ou dispositivos móveis que comunicam entre si. No entanto, ao contrário das redes públicas, as redes privadas requerem autorização prévia de alguma entidade (empresa ou pessoa) para se ligarem e fazerem parte deste tipo de rede. Normalmente, as redes de cadeias de bloqueio privadas são utilizadas em empresas ou corporações para realizar transacções ou operações com uma variedade de diferentes tipos de informação que podem ter um valor tangível sob a forma de documentos, processos, autorizações e/ou decisões empresariais aplicadas e supervisionadas por cadeias de bloqueio, exemplos: sector financeiro, sector de seguros, governo, entre outros.

3. O que é a programação Blockly?

Blockly é uma **linguagem de programação visual** composta por um simples conjunto de comandos que podemos combinar como se fossem as peças de um puzzle. É uma ferramenta muito útil para aqueles que querem **aprender a programar de forma** intuitiva e simples ou para aqueles que já sabem programar e querem ver o potencial deste tipo de programação.

Blockly é uma forma de programação em que não é necessário qualquer tipo de linguagem informática, isto porque é apenas juntar blocos gráficos como se estivéssemos a jogar lego ou um puzzle, só é preciso ter alguma lógica e mais nada!

Qualquer pessoa pode criar programas para telemóveis (smartphones) sem se meter com as linguagens de programação difíceis de compreender, basta juntar blocos de forma gráfica de uma forma simples, fácil e rápida de criar.

4. O que é o Termux?

Termux é um emulador de terminal Android e uma aplicação de ambiente Linux que funciona directamente sem a necessidade de encaminhamento ou configuração. É instalado automaticamente um sistema básico mínimo.

Utilizaremos Termux pela sua estabilidade e facilidade de instalação e gestão, no entanto, poderá utilizar um ambiente instalado do Ubuntu Linux para Android.

Neste ambiente Linux, terá o "núcleo" dos processos de comunicação da MiniBlocklyChain.

5. O que é a Mini BlocklyChain?

A Mini BlocklyChain é uma cadeia de bloqueio totalmente funcional é uma tecnologia desenvolvida para telemóveis com SO (Sistema Operativo) **Android** que serão os nós que executarão as transacções de envio e recepção. Criámos a primeira tecnologia de cadeia de blocos que é estruturada de forma "modular" através da programação de blocos onde qualquer pessoa com conhecimentos mínimos e sem saber programar pode criar e desenvolver programas para telemóveis e criar a sua própria cadeia de blocos, quer em modo de rede pública ou privada. Se quiser criar a sua própria moeda digital pode fazê-lo ou uma solução para a utilizar numa empresa, as possibilidades são baseadas nas necessidades de cada caso real e na lógica empresarial que o utilizador adopta ou cria de acordo com as suas necessidades.

A Mini BlocklyChain é a primeira cadeia modular de blocos para telemóveis onde um sistema transaccional pode ser implementado num curto espaço de tempo.

Antes de entrar na definição e utilização de blocos "módulo" precisamos de ter os conceitos básicos dos componentes da Mini BlocklyChain para saber quando, como e onde aplicá-los de acordo com o caso real que queremos implementar.

Os conceitos seguintes enfatizam o tipo de utilizador que estão a visar, pelo que não é importante para as pessoas que não têm conhecimentos de programação que os conceitos para os utilizadores de desenvolvimento sejam totalmente compreendidos.

Conceitos básicos:

Nó. - Cada dispositivo móvel (telefone, tablet, etc.) com um sistema operativo Android que faz parte da rede pública ou privada da Mini BlocklyChain é nomeado como um nó desta rede.

Haxixe. - É uma assinatura digital que está associada a um conjunto de dados, cadeia de caracteres, documentos ou algum tipo de informação digital, esta assinatura digital é única

e irrepetível, o que ajuda a enviar ou receber informação de modo a que o conteúdo inicial da informação enviada não possa ser alterado.

Activo. - Qualquer tipo de dados ou informações digitais que possam ser ponderados com algum valor tangível ou intangível para pessoas ou empresas (documentos, moedas digitais, música, vídeo, imagens, autorizações digitais, etc.).

Transacção. - Troca de informações entre nós que ocupam algum tipo de bem.

Transacção UXTO - É um tipo de transacção que embala uma ou várias transacções dos nós e todas as transacções não gasta de cada nó (UXTO: *Transaction Outputs não gasta*) podem ser gasta como entrada numa nova transacção. Estas transacções são agrupadas numa fila para serem processadas de cada vez que podem ser reguladas de acordo com os requisitos de cada fluxo de informação.

Transacção (input). - É um tipo de transacção que se baseia em transacções UXTO. Quando uma fila de transacções UXTO entra, é processada por uma transacção **de entrada** que classifica a transacção como um depósito ou uma despesa e assim poder ter um saldo do resultado final para cada utilizador.

Endereço de origem. - Este é o endereço da pessoa que gera ou envia a transacção a ser processada na fila do SQLite Master. É um número alfanumérico de 64 caracteres que é criado e verificado pelo sistema.

Endereço de destino. - Este é o endereço da pessoa que recebe a transacção e adiciona-o ao seu saldo ou armazena o bem enviado. É um número alfanumérico com 64 caracteres que é criado e verificado pelo sistema.

SQLite Master. - Base de dados API REST que recebe as transacções e cria uma fila para ser processada a cada determinado tempo variável ou fixo, de acordo com a necessidade do negócio.

Transacção DataBase de dados. - São as transacções que se reflectem na base de dados SQLite que reserva ou armazena informações sobre transacções de Mini BlocklyChain.

AES (Padrão Avançado de Criptografia) - É um processo de segurança que codifica os dados que são armazenados na base de dados SQLite do Mini BlocklyChain.

Equilíbrio. - Após efectuar qualquer processo de transacção no Mini BlocklyChain, obtém-se um balanço da operação quer como valor tangível (criptomoney) quer como valor intangível (processos comerciais entre pessoas ou empresas).

Processo empresarial. - É qualquer processo que envolva algum tipo de fluxo de informação para obter um resultado para um utilizador final, o utilizador final pode ser uma pessoa ou empresa de uma variedade de sectores do sector privado ou público.

Chave pública e privada. - É um tipo de codificação de informação em que duas chaves alfanuméricas associadas uma à outra são geradas e utilizadas para enviar informação sensível através de redes públicas ou privadas. A chave privada é utilizada para encriptar informação e quando a envia através da rede não pode ser alterada ou legível à primeira vista, a chave pública é aquela que é partilhada e é vista por qualquer pessoa ou empresa e esta ajudará a verificar a informação enviada, bem como a poder lê-la apenas pelo destinatário válido.

Assinatura digital. - É uma assinatura que pode ser criada com a chave privada, com esta assinatura o destinatário garante que a informação não foi alterada e confirma que a informação é válida para o destinatário.

Endereço digital (Mini BlocklyChain address). - É um endereço composto por caracteres alfanuméricos únicos para cada utilizador do Mini BlocklyChain, este endereço é utilizado para realizar transacções entre utilizadores e quer seja rede pública ou privada.

Número mágico. - Este é um número aleatório definido pelas regras de negócio para cada transacção UXTO em execução que serve para autorizar o nó a ser um candidato para executar a transacção UXTO e criar um novo bloco o Mini BlocklyChain.

Criação de um novo bloco. - Um novo bloco na Mini BlocklyChain é um hash criado e anexado pelo nó que saiu como o candidato vencedor para processar a transacção UXTO.

Protocolo de consenso PoW (Proof of Work). - É um teste que executa todos os nós e o primeiro nó que termina o teste com sucesso é o escolhido para executar a transacção UXTO. É um algoritmo que é composto de cálculos matemáticos para obter um resultado (hash) de acordo com as regras dadas em cada transacção executada pela Mini BlocklyChain baseia-se no desgaste ou procura dos recursos de processamento de informação dos computadores, no caso da Mini BlocklyChain foi estruturada e modificada para obter um "número mágico" este é um número para poder ter autorização ou consenso da maioria dos nós para ser aquele que pode executar a transacção UXTO. O PoW tem um nível máximo de dificuldade de 5, uma vez que só é utilizado para obter o "número mágico".

Protocolo de consenso PoQu (teste quântico) - É um teste que corre todos os nós e que é composto inicialmente pelo PoW para obter o "número mágico" e mais tarde executa um algoritmo de probabilidade baseado num QRNG (Quantum Random Number Generator) um gerador de números quânticos aleatórios, este processo assegura a igualdade de probabilidade para todos os nós e assim escolher o nó que executará a transacção UXTO e a criação do novo bloco.

Árvore Merkle. - Este é um método de segurança para assegurar a Mini BlocklyChain de que as transacções são válidas ou não foram modificadas por qualquer entidade externa. Uma árvore de hash merkle é uma estrutura de árvore de dados binária ou não binária na qual

cada nó que não é uma folha é marcado com o hash da concatenação das etiquetas ou valores dos seus nós filhos. São uma generalização de listas de hash e cadeias de hash.

Redis DB. - Base de dados de transacções em tempo real utilizada para processar e enviar para os nós as novas transacções que foram solicitadas.

SQLite DB. - Base de dados onde são armazenados os saldos e novos blocos para a Mini BlocklyChain para garantir a integridade da rede.

Sentinela. - Conector de segurança e integridade de dados entre Redis e SQLite. Este conector ou programa está encarregado de rever, processar, validar, distribuir e traduzir as transacções para os nós.

OpenSSH. - Conector de segurança para executar tarefas dentro do sistema operativo Android.

Terminal Termux Shell. - Programa onde pode encontrar dependências de terceiros para processar as transacções do Mini BlocklyChain, nesta versão 1.0 é utilizado para facilitar a instalação, contudo em versões futuras será substituído pelo sistema BlocklyShell que está actualmente em desenvolvimento.

Carteira. - É o repositório digital onde são mantidos dois dados fundamentais em qualquer transacção; as chaves públicas e privadas que serão utilizadas como endereço de origem e assinatura digital respectivamente para qualquer transacção realizada, bem como o saldo das transacções enviadas ou recebidas. É um repositório onde são mantidos os endereços do Mini BlocklyChain, bem como os resultados das transacções UXTO (Balanço).

Desenvolvedor ou programador de aplicações:

Programação Orientada a Objectos (Java) - A Mini BlocklyChain é feita em Java.

BlocklyCode. - É o termo dos contratos inteligentes que podem ser executados no ambiente do Mini BlocklyChain, o BlocklyCode é criado na programação java.

OpenJDK para Android. - É o conjunto de JDK e JRE para Android onde os BlocklyCode são criados e executados.

QRNG (Quantum Random Number Generator). - É um gerador de números quânticos aleatórios, Mini BlocklyChain tem actualmente dois APIs para a geração destes.

rsync. - É uma aplicação gratuita para sistemas do tipo Unix e Microsoft Windows que oferece uma transmissão eficiente de dados incrementais, que também funciona com dados comprimidos e encriptados.

ffsend. - É uma aplicação para partilhar ficheiros de forma fácil e segura a partir da linha de comando.

NTP. - Network Time Protocol, é um protocolo da Internet para sincronizar os relógios dos sistemas informáticos através do encaminhamento de pacotes em redes com latência variável.

Termux (desenvolvimento). - Terminal Shell com uma vasta gama de aplicações e bibliotecas de código aberto.

Módulos em bloco (dados). - Os desenvolvedores podem integrar módulos para melhorar as características do Mini BlocklyChain.

De um par para o outro. - Este termo é referido à comunicação entre nós de forma directa, ou seja, a actualização da informação não depende de um servidor central, mas cada nó funciona como um servidor central que comunica entre todos eles tendo a mesma informação, o que ajuda a evitar pontos de falha.

Sincthing. - ferramenta para sincronizar dados ou ficheiros entre dois dispositivos usando o tipo de comunicação "Peer to Peer".

Tor Vermelho. - Trata-se de uma rede de comunicações distribuídas de baixa latência sobreposta na Internet, na qual o encaminhamento de mensagens trocadas entre utilizadores não revela a sua identidade, ou seja, o seu endereço IP (anonimato em toda a rede).

Encaminhamento móvel. - Processo de instalação de software externo no seu telefone para entrar como administrador de sistema nos sistemas operativos Linux (Android) o utilizador administrador é chamado "root" para poder rodar o seu telefone terá acesso a qualquer processo. É importante notar que alguns fabricantes de telemóveis (Smartphones) dizem que a garantia é perdida devido a qualquer falha no telemóvel.

6. Arquitectura de processo em Mini BlocklyChain

A Mini BlocklyChain é formada por três processos que formam a sua arquitectura, o primeiro são os processos empresariais, o segundo são os processos de comunicação e o terceiro é o ambiente de desenvolvimento de módulos complementares e/ou a criação de "contratos inteligentes" de BlocklyCode.

Os processos empresariais são os blocos que formam uma série de rotinas para criar uma transacção quer em rede pública ou privada, este tipo de processo é o planeamento do negócio, o como, quando, o quê, quem, onde e mais atributos serão ordenados, planeados e distribuídos de modo a que a funcionalidade principal de cada transacção enviada, recebida, armazenada e/ou rejeitada seja alcançada. O primeiro processo é composto pelos seguintes tipos de blocos divididos em quatro categorias:

1. Blocos de entrada de dados
2. Blocos de processamento de dados
3. Blocos de segurança.
4. Blocos de dados modulares (desenvolvedores)
5. Bots de comunicação.

Os blocos de entrada de dados são aqueles que recebem a transacção com um mínimo de quatro parâmetros de entrada (**endereço de origem, endereço de destino, activo, dados**) e podem ter mais em função da variável "dados", pode ser um bloco desenvolvido por terceiros ou com um valor nulo (NULL).

Os blocos de processo de dados desenvolvem a logística, cálculo, normalização de dados, decisões lógicas e verificações de fluxo para cada transacção. Estes tipos de blocos são utilizados para dar inteligência ao processo empresarial e baseiam-se principalmente como o seu nome indica no processamento de todos os tipos de informação, bem como na sua conversão a partir de diferentes tipos de dados.

Os blocos de segurança são utilizados para validar a informação e assegurar que as transacções não foram alteradas desde a sua origem até ao seu destino, são sempre processados com vários algoritmos de segurança utilizados nas tecnologias de cadeia de bloqueio, entre as ferramentas mais utilizadas estão (assinatura hash, assinatura digital, encriptação de dados AES, criação e validação de endereços digitais, entre outras).

Os blocos de dados modulares, estes são os blocos que são desenvolvidos por terceiros, lembrem-se que a Mini BlocklyChain foi criada de uma forma modular para ser enriquecida por novos blocos de acordo com as necessidades de cada sector, público ou privado. Para saber mais sobre como criar módulos, consulte o Anexo dos Desenvolvedores.

Como comentamos anteriormente a arquitectura do Mini BlocklyChain na sua segunda componente são os processos de comunicação, estes processos são os encarregados dos canais de comunicação via TCP e Sockets de comunicação para dar flexibilidade de envio e recepção de transacções quer pelos diferentes meios que são utilizados no momento mais utilizados: Internet móvel, Wifi a trabalhar actualmente para incluir o meio de comunicação Bluetooth.

Os blocos de comunicação baseiam-se basicamente na troca de dados com segurança implementada no canal de transferência e esta baseia-se numa comunicação através do protocolo SSH (Secure Shell) com as diferentes fases pelas quais passa uma transacção enviada ou recebida.

A parte das comunicações é fundamental uma vez que estes processos têm a função de actualizar a informação em todos os nós via TCP e a ligação chamada "**Peer to Peer**" os blocos que intervêm na comunicação baseiam-se na troca de informação entre os nós sem a intervenção de servidores intermediários de modo a que cada nó o torne independente podendo criar uma rede de nós onde os pontos de falha são mínimos ou quase nulos. Do mesmo modo, esta independência de cada nó ajuda-os a tomar decisões individualmente ou como um todo, de acordo com as necessidades do negócio.

A arquitectura "Peer to Peer" consiste em três partes que formam a rede pública ou privada que se decide criar, em ambos os casos o canal de comunicação é encriptado de nó para nó.

O primeiro componente de comunicação entre dispositivos móveis (smartphones) ou wifi é fornecer aos nós ou dispositivos uma rede onde possam ser encontrados em qualquer parte do mundo, a rede de comunicação onde MiniBlocklyChain está montada é a rede "**Tor**".

O que é a rede Tor? - (<https://www.torproject.org>)

"**Tor** (acrónimo de Te **Onion Router** - em espanhol) é um projecto cujo principal objectivo é o desenvolvimento de uma rede de comunicações distribuídas de baixa latência sobreposta na Internet, em que o encaminhamento de mensagens trocadas entre utilizadores não revela a sua identidade, ou seja, o seu endereço IP (anonimato ao nível da rede) e que, além disso, mantém a integridade e o sigilo da informação que viaja através dela".

A segunda componente e não menos importante tarefa é conseguir que todos os nós em MiniBlocklyChain tenham os mesmos dados em qualquer altura ou tenham as suas bases de dados e ficheiros sincronizados para realizar esta tarefa entre os nós serão implementados "**syncing**".

O que é a rede Syncing? - (<https://syncing.net>)

Syncing é uma aplicação gratuita de sincronização de ficheiros peer-to-peer de código aberto disponível para Windows, Mac, Linux, Android, Solaris, Darwin e BSD. Pode sincronizar ficheiros entre dispositivos numa rede local ou entre dispositivos remotos através da Internet.

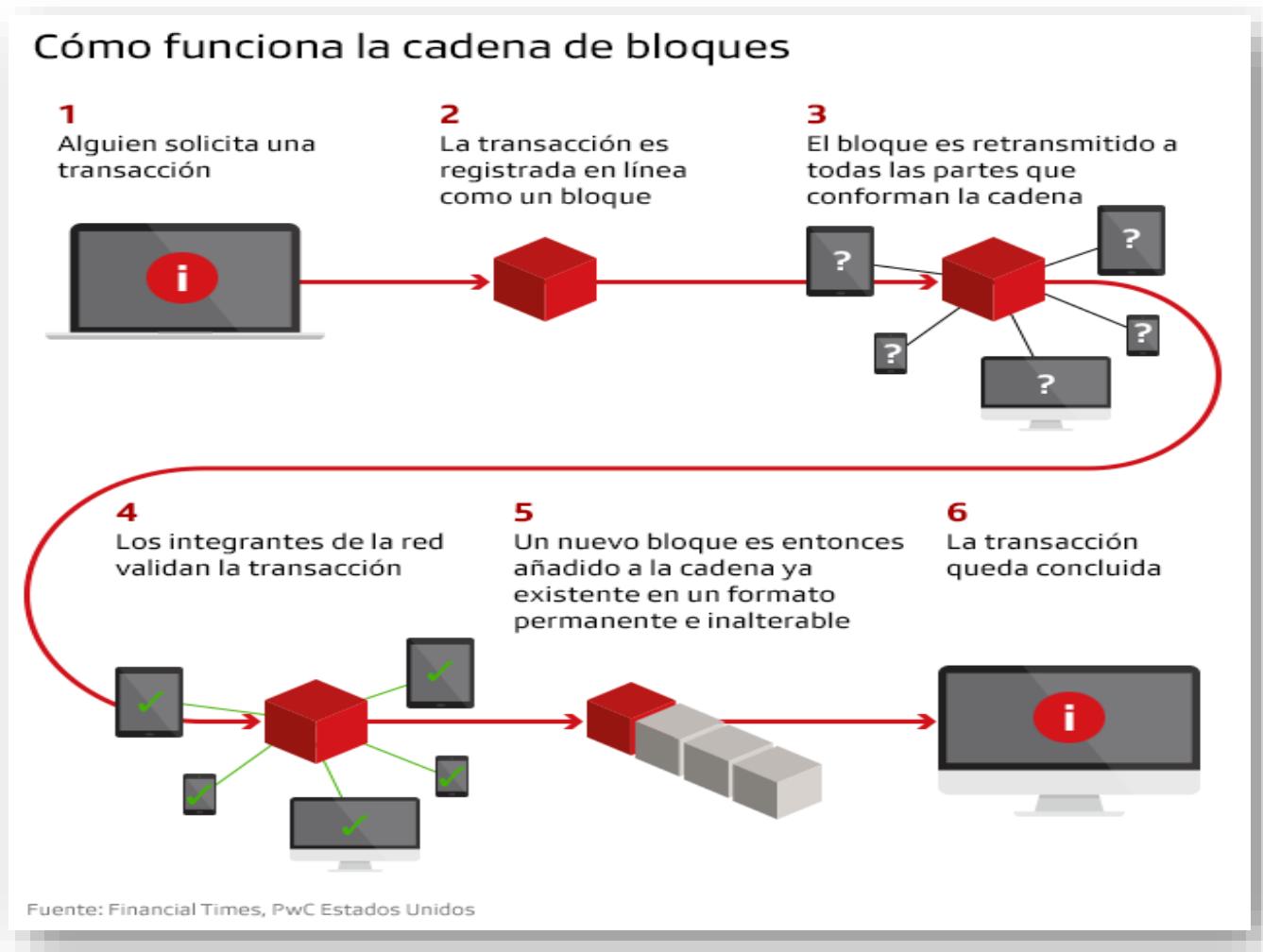
Com base nos dois componentes de comunicação anteriores, podemos começar a desenvolver uma rede de confiança entre nós e com o nível de sincronização de dados que será a espinha dorsal ou "núcleo" dos processos empresariais, a fim de satisfazer todas as necessidades do utilizador ou da empresa.

O terceiro componente da rede de comunicações é a base de dados Redis, que notificará todos os nós em tempo real de novas transacções que são recebidas e de novas transacções que são enviadas.

O que é a rede Redis DB? - (<https://redis.io>)

Redis é um motor de base de dados na memória, baseado no armazenamento em tabelas de hash (chave/valor) mas que pode opcionalmente ser utilizado como uma base de dados durável ou persistente.

7. Diagrama de funcionalidad BlocklyChain (Mini BlocklyChain).



8. O que é o Mini BlocklyCode?

Mini BlocklyCode são programas criados na linguagem Java e são armazenados num repositório independente dos nós, são normalmente nomeados como contratos inteligentes, estes programas são concebidos com condições lógicas para que, quando estas condições são cumpridas, sejam executados automaticamente sem depender de qualquer autorização ou intervenção humana externa.

"Um contrato inteligente é um programa informático que facilita, assegura, aplica e executa acordos registados entre duas ou mais partes (por exemplo, indivíduos ou organizações). Como tal, ajudá-los-iam na negociação e definição de tais acordos que levarão à realização de determinadas acções em resultado do cumprimento de uma série de condições específicas.

Um contrato inteligente é um programa que vive num sistema não controlado por nenhuma das partes, ou pelos seus agentes, e que executa um contrato automático que funciona como uma sentença "se" de qualquer outro programa de computador. Com a diferença de que é feito de uma forma que interage com bens reais. Quando uma condição pré-programada, não sujeita a qualquer julgamento humano, é desencadeada, o contrato inteligente executa a cláusula contratual correspondente.

O seu objectivo é proporcionar maior segurança do que o direito contratual tradicional e reduzir os custos de transacção associados à contratação. A transferência de valor digital através de um sistema que não requer confiança (por exemplo, bitcoins) abre a porta a novas aplicações que podem fazer uso de contratos inteligentes. "

O Mini BlocklyCode é destinado a programadores com experiência em programação java. No Mini BlocklyChain alguns tipos de bibliotecas são restritas para segurança do mesmo sistema, contudo, podem ser criadas bibliotecas para criar transacções para enviar ou receber algum valor contratual criado ou acordado por duas ou mais partes.

Cada Mini BlocklyChain está em conformidade com as seguintes premissas:

- ✓ Qualquer Mini BlocklyChain criado é baseado na execução apenas de mensagens e não de execuções no sistema, no entanto, estas mensagens podem conter autorizações, aprovações de contratos físicos ou acções físicas.
- ✓ Cada Mini BlocklyChain criado tem de passar pelo bloco de comunicação "auditor". Este bloco é responsável pelo controlo de código malicioso ou código proibido para rever sentenças ou ordens não autorizadas no sistema.
- ✓ Todos os Mini BlocklyChain são executados apenas na JVM do nó fonte, os programas não são executados globalmente para protecção do sistema.

Para mais detalhes ver o apêndice "Mini BlocklyChain for Developers".

9. Instalação de rede de comunicações Mini BlocklyChain

1. Instalação e configuração da rede Mini SQLSync

A rede Mini SQLSync está encarregada de receber as transacções dos nós para que possam processar essas transacções. Esta rede é formada por uma rede principal que é através de "Peer to Peer" entre os nós e uma rede de backup chamada Mini Sentinel RESTful.

Começaremos com a instalação da rede de backup RESTful Mini Sentinel, este serviço é o que receberá as transacções dos nós, este serviço baseia-se no armazenamento das transacções durante um determinado tempo para mais tarde converter a informação através de um conector que injectará uma fila de todas as transacções encriptadas para um serviço Redis. Mais tarde, o serviço de base de dados Redis executará em tempo real a libertação da fila de transacções encriptadas para todos os nós que integram a rede Mini BlocklyChain.

Um serviço ou desenho do tipo RESTful é uma forma simples e fácil de consultar, modificar, apagar e/ou inserir informação numa base de dados através de um URL ou endereço de Internet, no nosso caso utilizaremos a base de dados SQLite devido às suas características de ser toda a informação num único ficheiro. Para uma utilização de requisitos com necessidades de grande escala poderemos utilizar outro tipo de base de dados como MySQL, Oracle, DB2 ou outra base de dados comercial, teremos simplesmente de mudar o conector da Mini BlocklyChain do SQLite (versão gratuita) para o conector da Mini BlocklyChain DB outros (versão comercial).

NOTA IMPORTANTE: A configuração RESTful Mini Sentinel não processa qualquer tipo de transacção em qualquer altura, é apenas o serviço que forma "a informação" para que os nós possam realizar o processamento da transacção correctamente e num tempo planeado e sincronizado para todos os nós.

A instalação do serviço RESTful é baseada numa base de dados SQLite. Esta instalação será feita em ambiente Windows para fins práticos, contudo, este modelo pode ser facilmente replicado num sistema operativo do tipo Linux.

Para aqueles que desejam rever o desempenho e apoio de consultas e inserções SQLite, por favor consultem estes testes de desempenho:

<https://www.sami-lehtinen.net/blog/sqlite3-performance-testing>

<https://sqlite.org/src4/doc/trunk/www/lsmperf.wiki>

Instalação de rede de apoio RESTful Mini Sentinel. Iremos instalar e configurar este serviço num computador Windows 10, no entanto, o processo também foi facilmente instalado num ambiente Linux Ubuntu 18.09.

Requisitos:

- ✓ Servidor de base de dados SQLite em modo RESTful (<https://github.com/olsonpm/sqlite-to-rest>).
- ✓ Conector SQLite-Redis Sentinel
- ✓ Servidor de base de dados Redis em modo Master-Slave (<https://redis.io/topics/replication>)

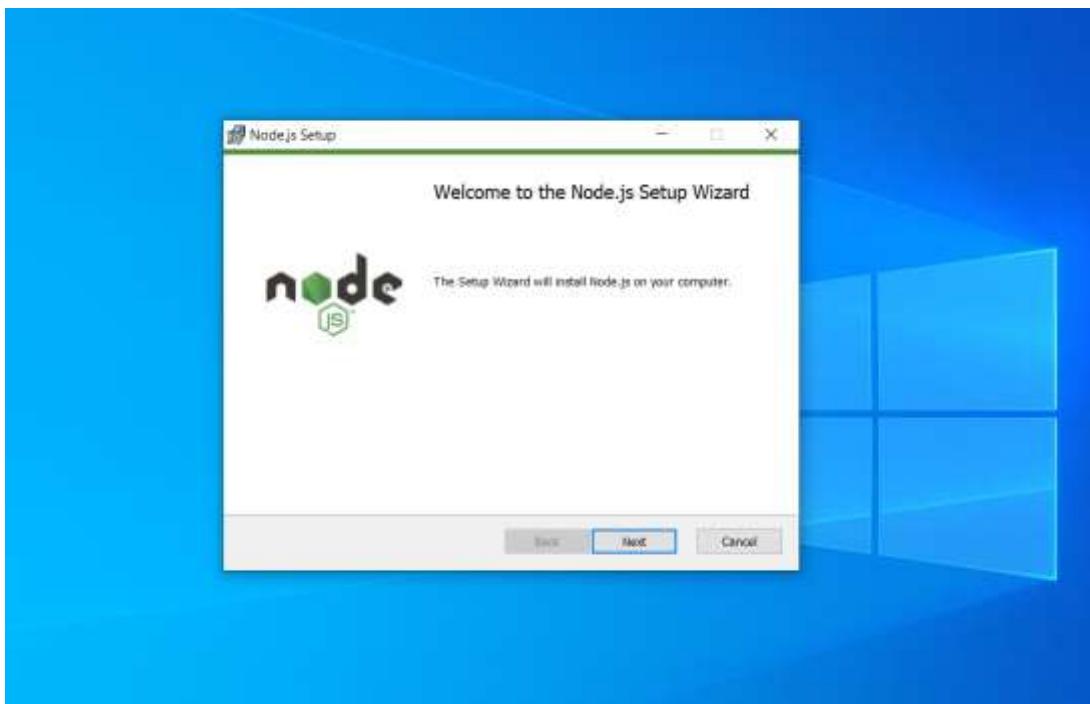
Instalação do servidor de base de dados SQLite em modo RESTful

Para a instalação precisamos de começar com os seguintes pacotes de software, Nodejs, sqlite3 e Git Bash para Windows.

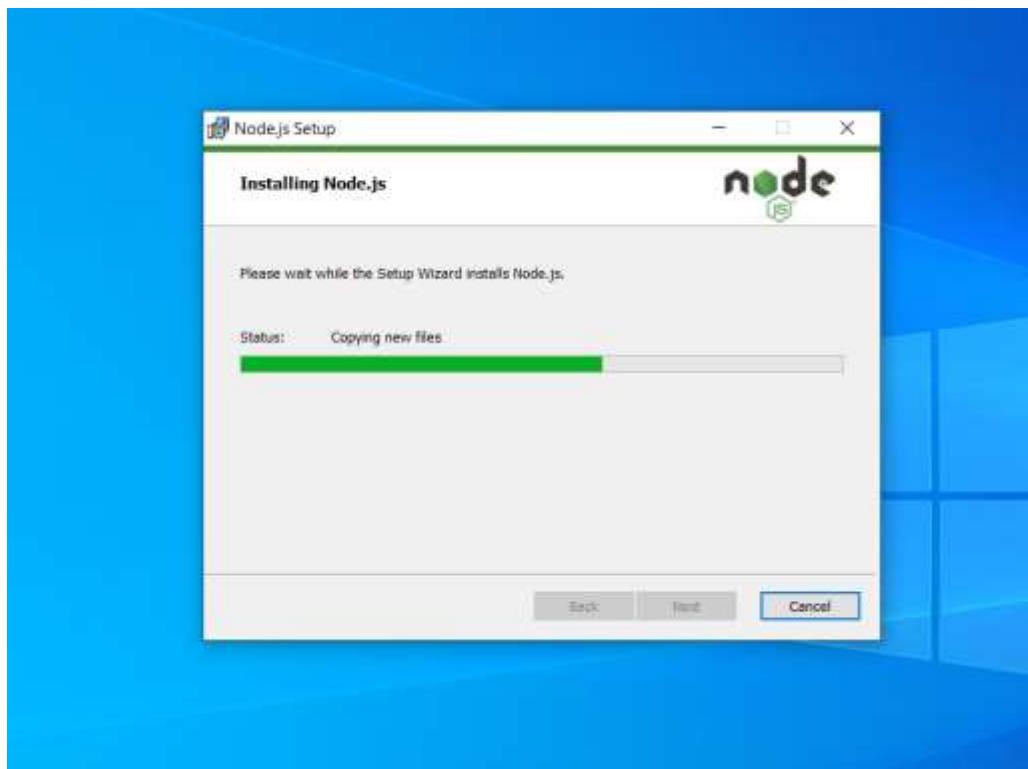
Para obter o Nodejs, consultámos o seu sitio oficial <https://nodejs.org/es/> e escolhemos a versão "Recomendado para a maioria".

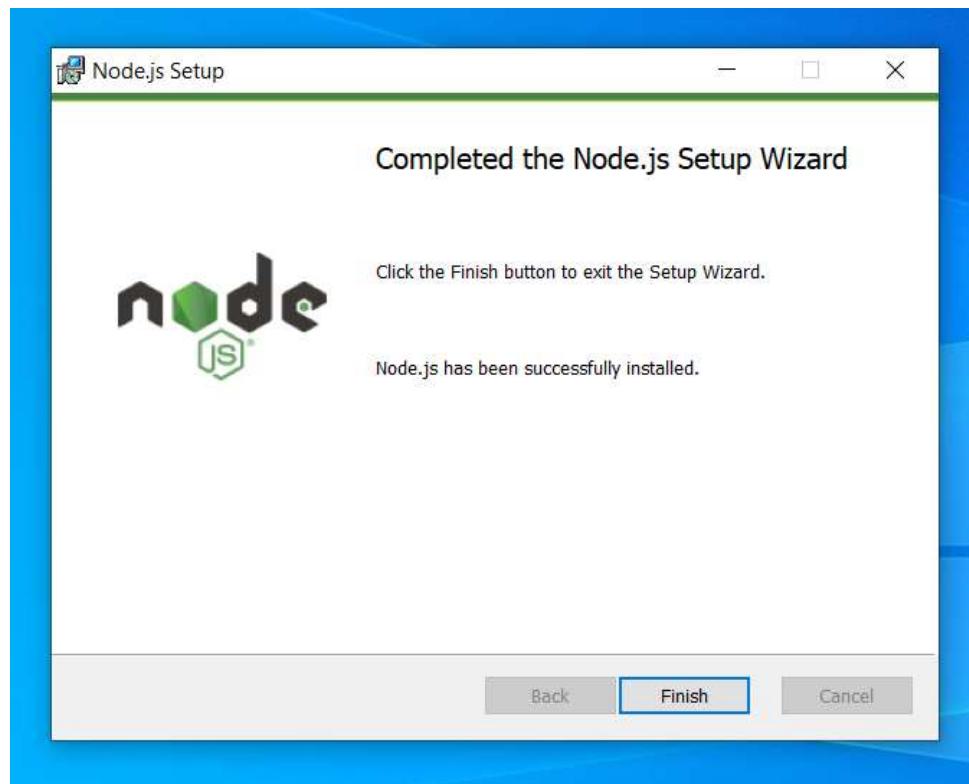


Depois de descarregar o ficheiro com a extensão .msi, fazemos duplo clique para o instalar.



Realizamos a instalação por defeito, basta clicar em "Next" (Seguinte) sem escolher qualquer opção adicional que nos peça.



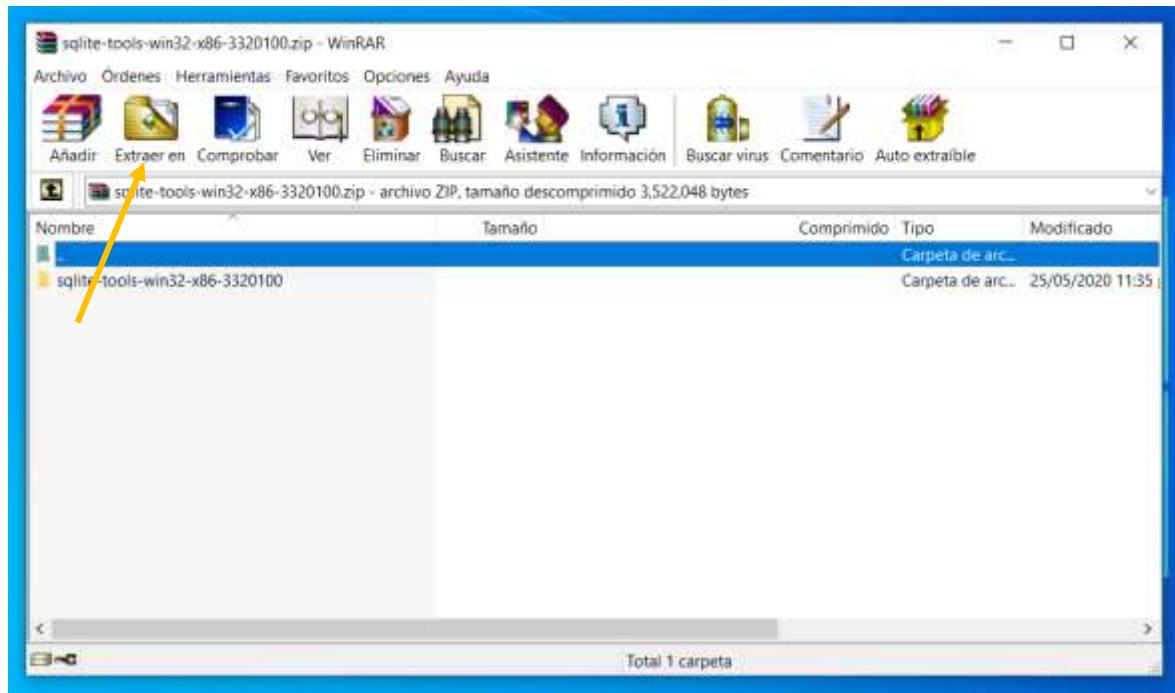


Desde que terminámos a instalação de Nodejs, continuamos com a instalação da base de dados SQLite.

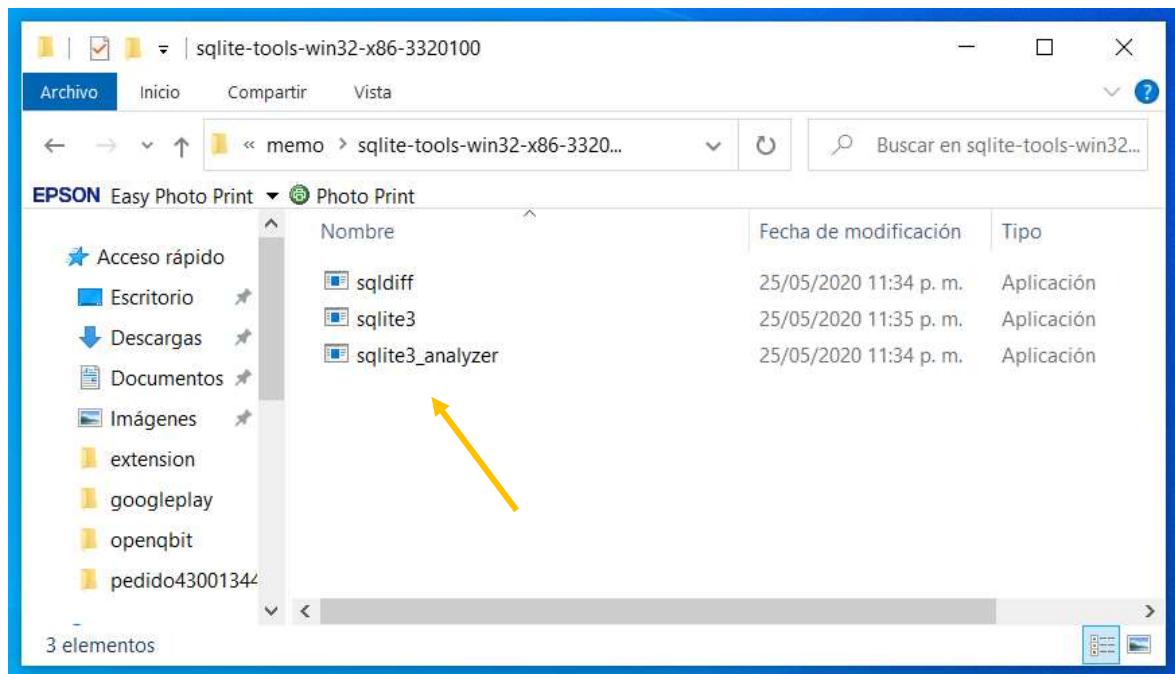
Vamos ao seu site oficial, <https://www.sqlite.org/index.html> e clicamos na parte onde "descarrega".

A screenshot of the SQLite website. The URL in the browser bar is https://www.sqlite.org/index.html. A yellow arrow points from the text "descarrega" in the previous paragraph to the "Download" button in the navigation menu. The menu includes links for Home, About, Documentation, Download (which is highlighted), License, Support, and Purchase. Below the menu, there's a section titled "What Is SQLite?" with a brief description and a link to "More Information...". Another section discusses the SQLite file format and its widespread use. At the bottom, there's a "Latest Release" section with a link to Version 3.32.1 (2020-05-25) and download links for "Download" and "Prior Releases".

De seguida escolhemos a opção onde diz "Binários Pré-compilados para Windows" e clicamos na versão de software "sqlite-tools-win32-x86-3320100.zip" quando o download estiver terminado procedemos à descompressão do ficheiro de forma simples, abrimos a pasta onde o ficheiro foi descarregado e fazemos duplo clique no ficheiro para o descomprimir.

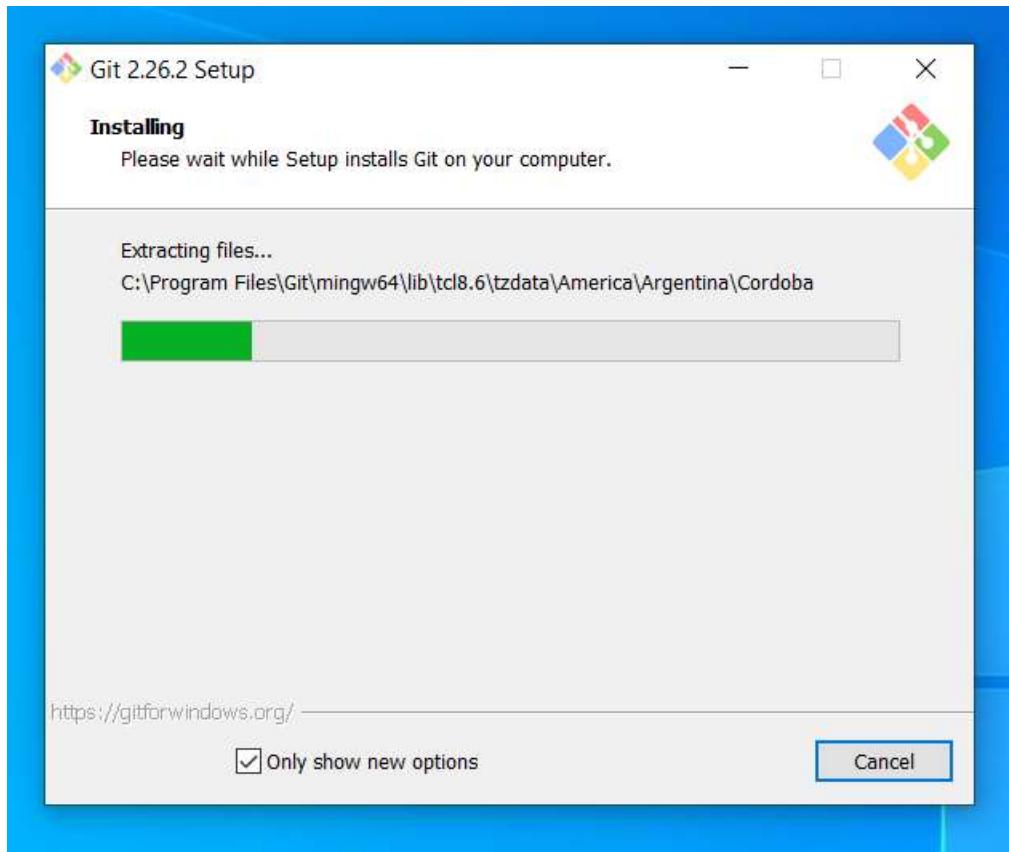


Após a descompressão, teremos três ficheiros, estes são o nosso ambiente para criar a nossa base de dados SQLite que utilizaremos mais tarde.



Vamos começar por instalar o Git Bash, um emulador de terminal semelhante ao Linux que nos ajudará a executar correctamente o serviço de backup RESTful.

Descarregá-lo-emos do seu site oficial, <https://gitforwindows.org/> e procederemos à sua instalação com as opções padrão que indica.



Agora que temos nodejs, sqlite e Git Bash instalados na nossa máquina Windows 10, procedemos à instalação do ambiente que suportará a nossa base de dados SQLite em modo RESTful.

No momento do download do software que dará a funcionalidade de RESTful ao SQLite. Para isso temos de ir ao seguinte site, <https://github.com/olsonpm/sqlite-to-rest> neste clicaremos no botão verde direito "Clone ou download" e escolheremos a opção "Download ZIP" isto irá descarregar o software no nosso computador.

No description or website provided.

automatic-api

Branch: dev • New pull request

Find file • Clone or download

olsonpm add prettier and eslint

- bin add prettier and eslint
- cli/commands add prettier and eslint
- docs add prettier and eslint
- lib add prettier and eslint
- tests add prettier and eslint
- .gitignore add prettier and eslint

Clone with HTTPS

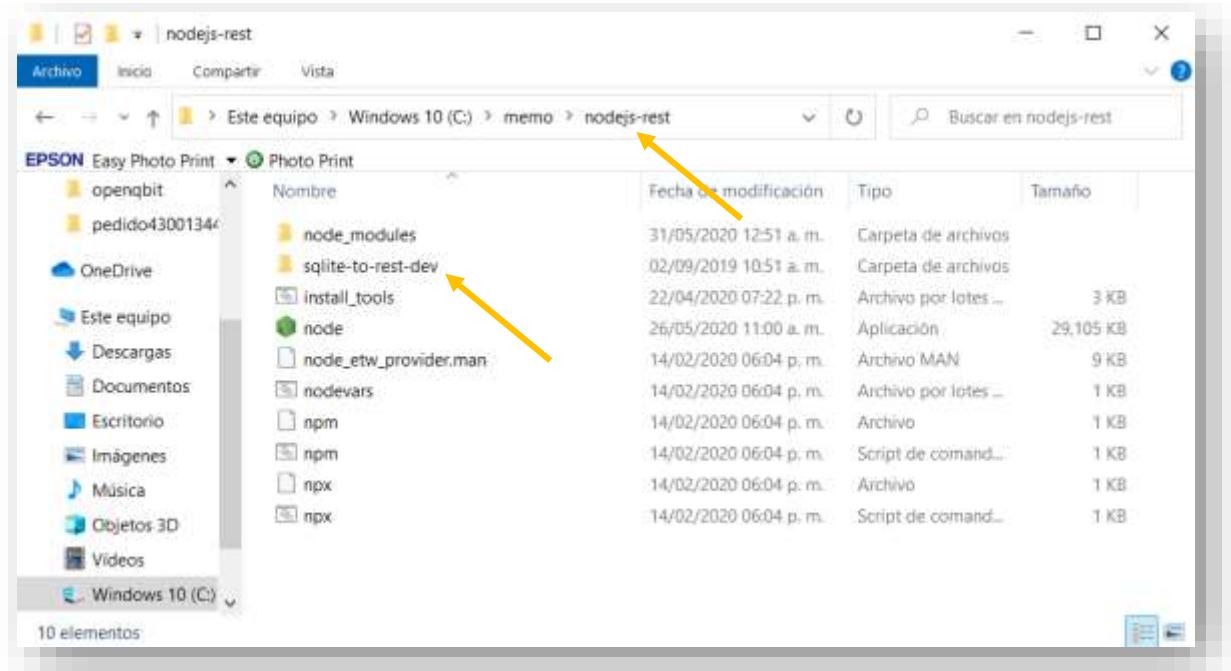
Use Git or checkout with SVN using the web URL.
https://github.com/olsonpm/sqlite-to-rest

Open in Desktop • Download ZIP

9 months ago • 9 months ago • 9 months ago

Após descarregá-lo no nosso computador, procedemos à sua descompressão dentro do directório ou pasta onde instalámos o programa **nodejs** e teremos um directório com o nome "**sqlite-to-rest-dev**".

NOTA: É importante que o directório **sqlite-to-rest-dev** esteja dentro do directório onde **nodejs** foi instalado.



Tendo tudo instalado, prosseguimos com a configuração da base de dados SQLite que utilizaremos para armazenar as transacções dos nós no ambiente de backup RESTful.

Desenho de tabelas e estrutura de dados. Comandos a serem executados online na lista de comandos CMD

```
sqlite3 op.sqlite3 "CREATE TABLE trans (id integer primary key, addro, addrd, value);"
```

```
sqlite3 op.sqlite3 "CREATE TABLE sign (id integer primary key, trans_id references trans (id), sign, hash);"
```

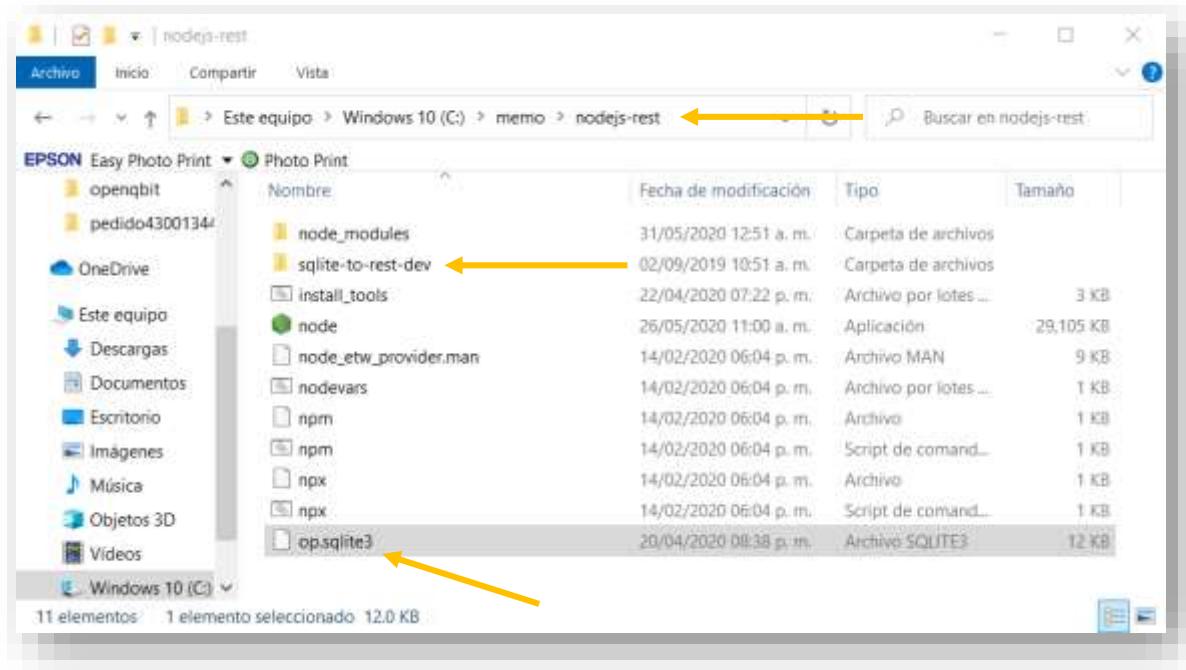
```
C:\memo>sqlite3 op.sqlite3 "CREATE TABLE trans(id integer primary key, addro, addrd, value);"
C:\memo>sqlite3 op.sqlite3 "CREATE TABLE sign(id integer primary key, trans_id references trans (id), sign, hash);"
C:\memo>sqlite-tools-win32-x86-3320100>dir
El volumen de la unidad C es Windows 10
El número de serie del volumen es: E019-5C05

Directorio de C:\memo>sqlite-tools-win32-x86-3320100

31/05/2020 02:29 a. m.    <DIR>        .
31/05/2020 02:29 a. m.    <DIR>        ..
31/05/2020 02:29 a. m.      12,288 op.sqlite3
25/05/2020 11:34 p. m.     516,608 sqlcipher.exe
25/05/2020 11:35 p. m.     972,800 sqlite3.exe
25/05/2020 11:34 p. m.     2,032,640 sqlite3_analyzer.exe
4 archivos            3,534,336 bytes
2 dirs   137,272,078,336 bytes libres

C:\memo>sqlite-tools-win32-x86-3320100>
```

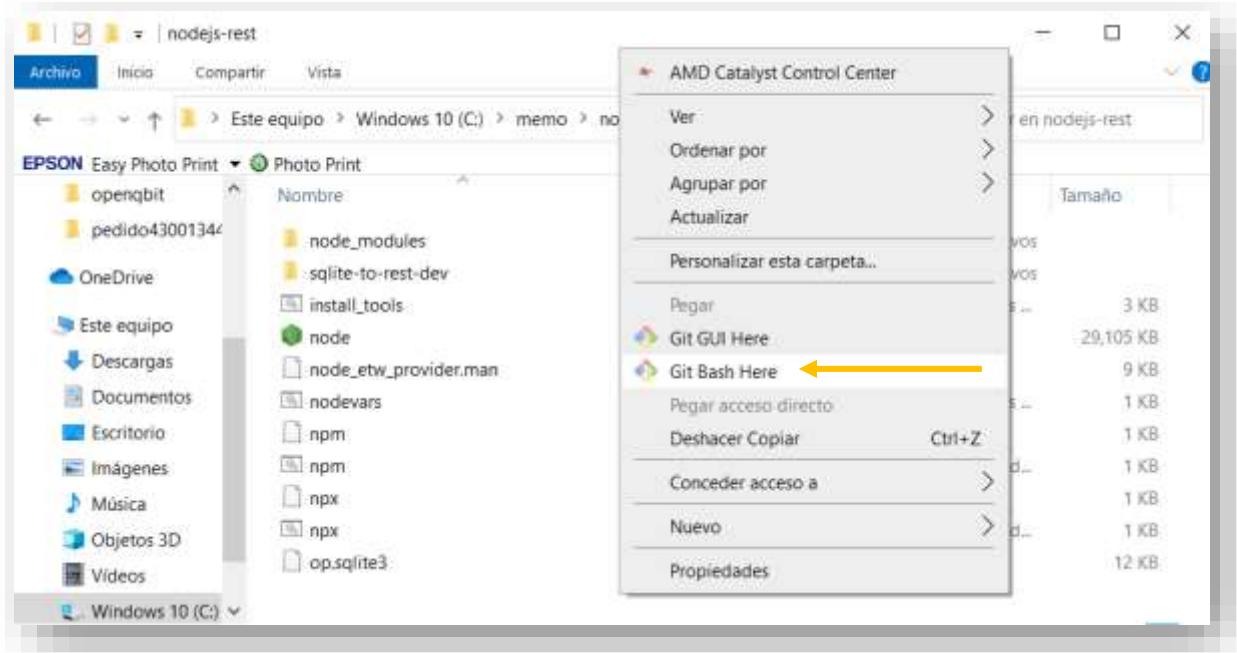
Após a criação da base de dados op.sqlite3, devemos fazer uma cópia no directório onde o nodejs foi instalado. No directório nodejs deve também estar a cópia de "sqlite-to-rest-dev".

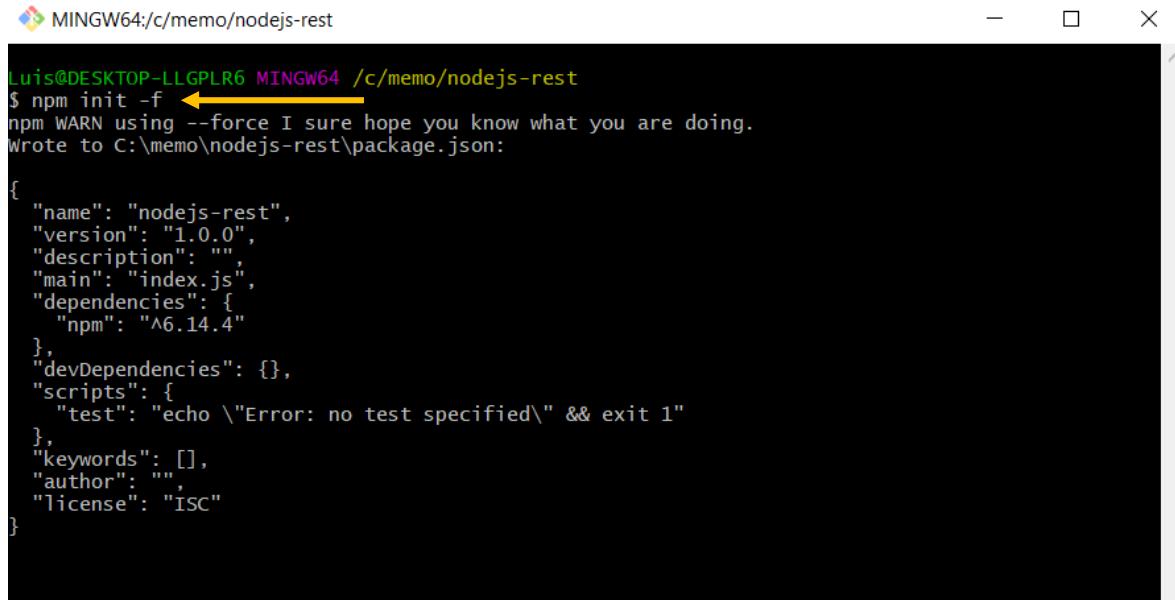


Colocamo-nos na pasta onde está a instalação dos **nodejs** e onde o software "**sqlite-to-rest-dev**" também deve estar. Aponte para a pasta com o ponteiro e clique com o botão direito do rato para exibir o menu e escolha onde diz "**Git Bash**" para abrir um terminal.

No novo terminal aberto Git Bash, executamos os seguintes comandos:

```
$ npm init -f
```

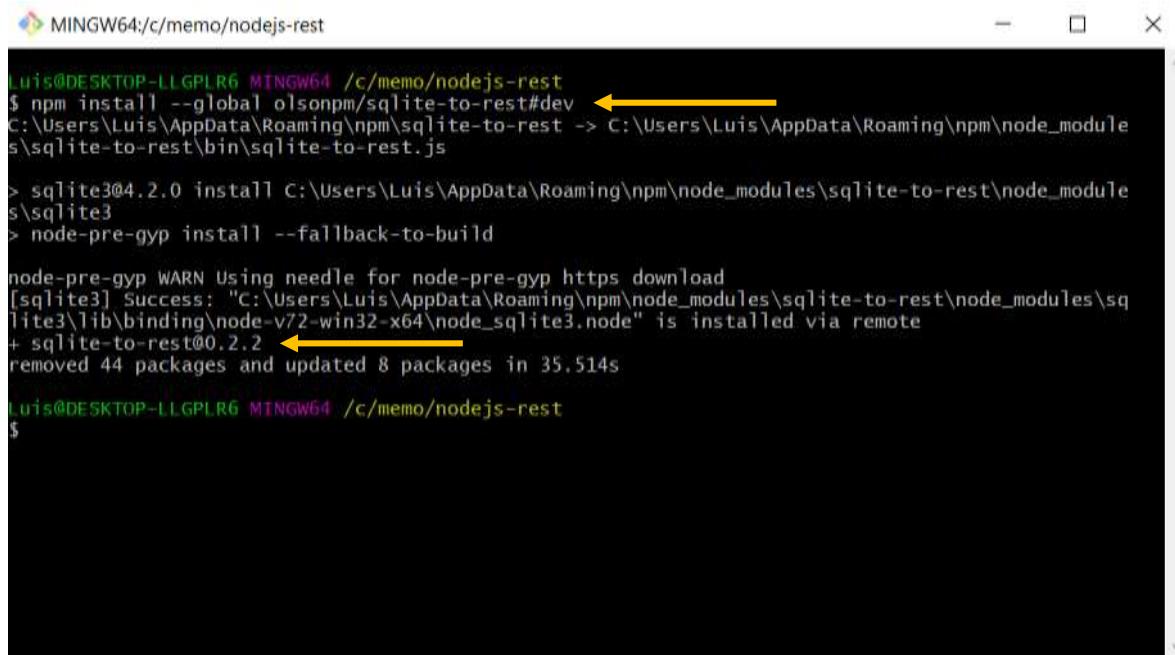




```
Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$ npm init -f
npm WARN using --force I sure hope you know what you are doing.
Wrote to C:\memo\nodejs-rest\package.json:

{
  "name": "nodejs-rest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "dependencies": {
    "npm": "^6.14.4"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

\$ npm install -- globo olsonpm/sqlite-to-rest#dev



```
Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$ npm install --global olsonpm/sqlite-to-rest#dev
C:\Users\Luis\AppData\Roaming\npm\sqlite-to-rest -> C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\bin\sqlite-to-rest.js

> sqlite3@4.2.0 install C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\node_modules\sqlite3
> node-pre-gyp install --fallback-to-build

node-pre-gyp WARN Using needle for node-pre-gyp https download
[sqlite3] Success: "C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\node_modules\sqlite3\lib\binding\node-v72-win32-x64\node_sqlite3.node" is installed via remote
+ sqlite-to-rest@0.2.2
removed 44 packages and updated 8 packages in 35.514s

Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$
```

Após a execução do comando, aparecerá a instalação do pacote "sqlite-to-rest".

Gerámos o ambiente RESTful para SQLite com a base **op.sqlite3** que criámos anteriormente.

Executamos o comando: `$ sqlite-para-junta gerar-esqueleto --db-path ./op.sqlite3`



```
MINGW64/c/memo/nodejs-rest
u150DESKTOP-LLGPLRS MINGW64 /c/memo/nodejs-rest
sqlite-to-rest generate-skeleton --db-path ./op.sqlite3
package.json found in working directory.
Installing dependencies.
Writing the skeleton server to: skeleton.js
finished!
u150DESKTOP-LLGPLRS MINGW64 /c/memo/nodejs-rest
```

Iniciamos o serviço RESTful SQLite na porta padrão 8085. Executamos o seguinte comando:
`$ node skeleton.js`



```
MINGW64/c/memo/nodejs-rest
u150DESKTOP-LLGPLRS MINGW64 /c/memo/nodejs-rest
node skeleton.js
listening on port: 8085
```

Testamos o serviço RESTful com a adição de dados e consulta de dados.



```
MINGW64/c/memo/nodejs-rest
u150DESKTOP-LLGPLRS MINGW64 /c/memo/nodejs-rest
curl -s -H "Content-Type: application/json" -d '{"id":6,"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}' http://localhost:8085/trans
u150DESKTOP-LLGPLRS MINGW64 /c/memo/nodejs-rest
curl -s http://localhost:8085/trans

[{"id":1,"addr": "CO","addrd": "soulder","value": "Avery"}, {"id":2,"addr": "WI","addrd": "New Glarus","value": "New Glarus"}, {"id":3,"addr": "WI","addrd": "Madison","value": "One Barrel"}, {"id":4,"addr": "WI","addrd": "Madison","value": "One Barrel"}, {"id":5,"addr": "WI","addrd": "Madison","value": "One Barrel"}, {"id":6,"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}]
```

Para testar o serviço SQLite RESTful, usamos os seguintes comandos:

Para inserir dados na tabela trans que se encontra dentro da base de dados op.sqlite3:

```
$ curl -s -H "Content-Type: application/json" -d '{"addro":"QWERTY1234", "addrd":"ASDFG4567", "value": "999"}' http://localhost:8085/trans
```

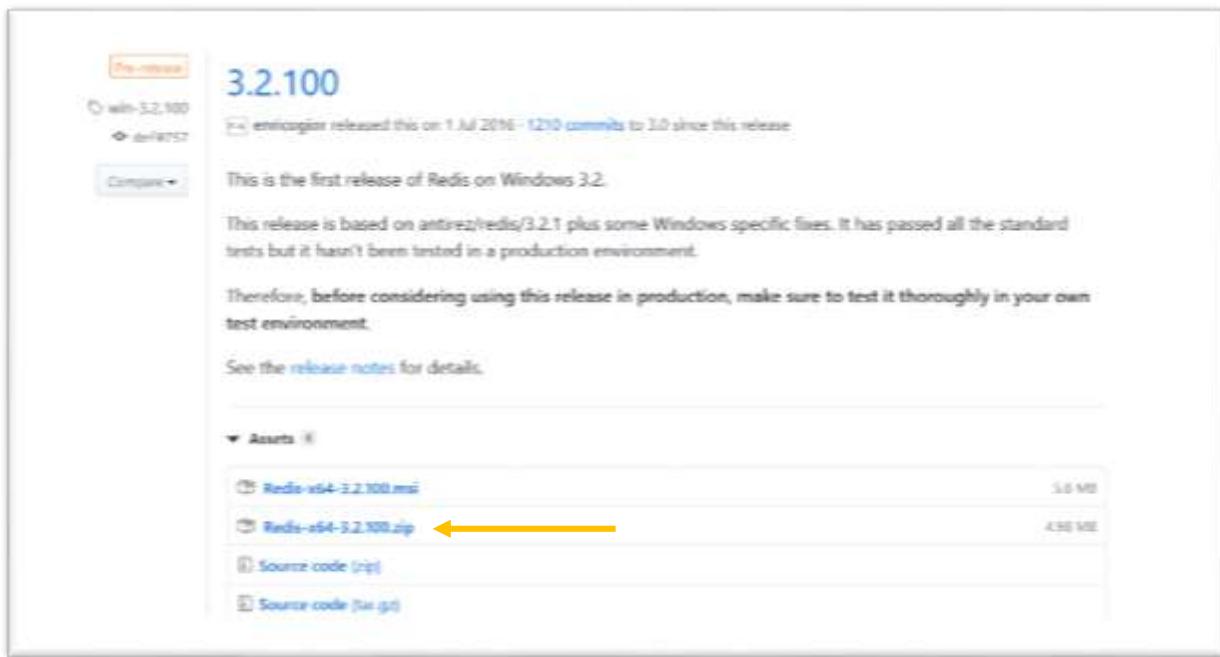
Para fazer uma consulta de todos os dados da tabela trans:

```
$ encaracolar -s -H 'intervalo: filas=0-2' http://localhost:8085/trans
```

Para rever como utilizar em detalhe todos os serviços RESTful habilitados (consultar, inserir, actualizar e/ou apagar dados) rever o [Apêndice "Restful SQLite GET/POST comandos"](#).

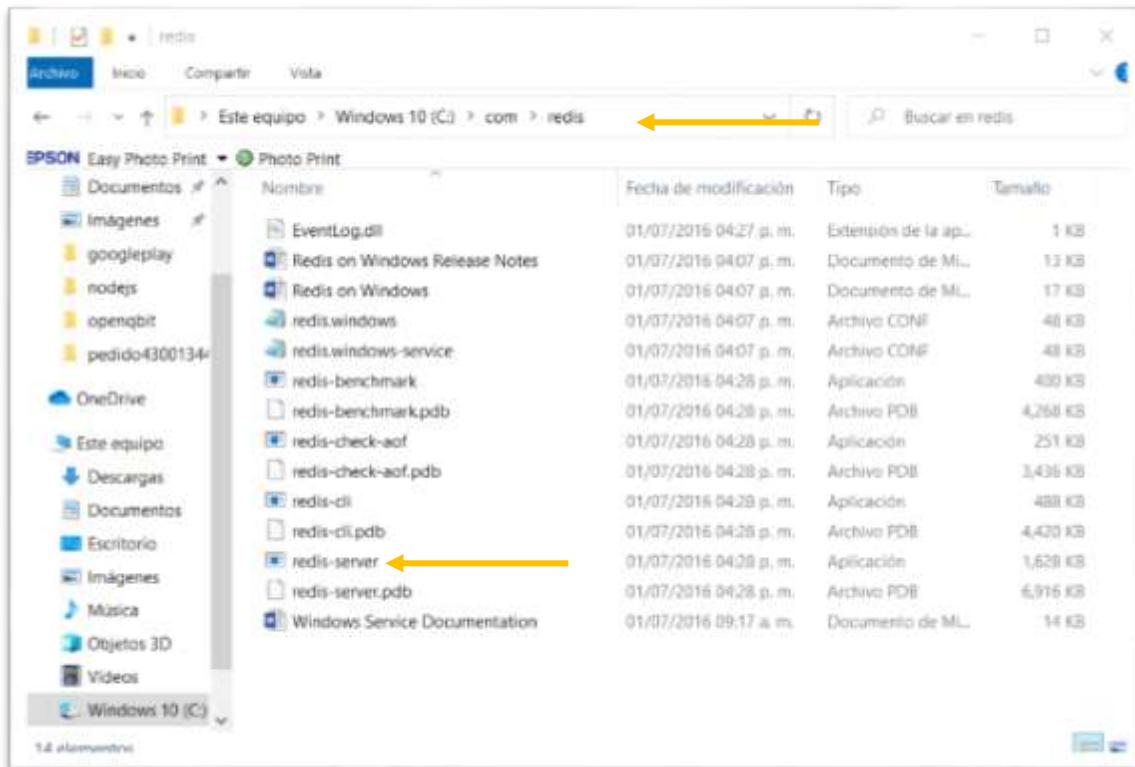
NOTA: Na instalação anterior, todas as consultas estão a ser feitas no URL com o endereço "localhost", contudo, quando o servidor é exposto com um IP público (Internet) ou IP privado (Wifi), funcionará sem qualquer problema. Vamos testar isto quando estivermos a fazer os testes de comunicação entre o serviço SQLite RESTful e os nós da rede de comunicação que formam a Mini BlocklyChain.

Instalação da base de dados redis para serviço de rede de backup, para descarregar o software redis para Windows temos de ir ao site, <https://github.com/microsoftarchive/redis/releases/tag/win-3.2.100> e escolher o pacote ZIP.



Para localizar a instalação vamos criar um directório chamado "redis" dentro do Windows e descarregar o ficheiro com uma extensão ZIP, descomprimimo-lo no directório criado anteriormente, já temos a instalação concluída redis.

Testamos a instalação executando o servidor com um duplo clique no comando "redis-server".



Depois de executar o comando "redis-server" veremos o servidor a correr num terminal de comando CMD do Windows na porta por defeito 6379:

```
C:\com\redis>redis-server.exe
[10192] 31 May 13:50:45.340 # Warning: no config file specified, using the default config. In order to specify a config
File use C:\com\redis\redis-server.exe /path/to/redis.conf

Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 10192

http://redis.io

[10192] 31 May 13:50:45.355 # Server started, Redis version 3.2.100
[10192] 31 May 13:50:45.355 * The server is now ready to accept connections on port 6379
```

Neste teste temos uma versão do Redis 3.2.100 para Windows 10, no caso do sistema operativo Linux temos a versão 6.0.4 (lançada em Maio de 2020) no entanto para a nossa OpenQbit.com

configuração do Mini BlocklyChain a versão Windows tem características de funcionalidade suficientes de que necessitamos.

Para parar a execução, fazemos a combinação chave Ctrl + C. Procedemos à configuração da base de dados Redis 3.2.100 para Windows 10 com uma configuração entre Redis e os nós da rede de comunicações Mini SQLSync, o tipo **Master(Master)-Slave(Slave)** é distribuído da seguinte forma:

O Master é o servidor Redis para Windows 10 que estamos a configurar e este irá replicar os dados em tempo real e sincronizados com a mesma informação para todos os nós (telemóveis). Estes nós terão um servidor Redis instalado em modo **Slave**.

Nesta altura, começamos a ver como funciona a rede de apoio que estamos a instalar e a configurar. Esta configuração servir-nos-á para comunicar com todos os nós em tempo real, ajudar-nos-á a comunicar com todos os nós da rede quando houver uma nova fila de transacção a ser processada pelos nós, numa fila de igualdade na transferência de informação para todos os nós, para que qualquer nó tenha a mesma probabilidade de poder processar a informação da "fila de transacção".

Iniciamos a configuração do conector **SQLite-Redis Sentinel**.

Este conector é um programa desenvolvido em linguagem Java e como o seu nome indica, liga as bases de dados SQLite e Redis (**Master**).

A função do conector é transmitir a informação (transacções) de SQLite para Redis (**Mestre**) e isto envia a "fila de transacções" para os nós (**Escravos**).

Para mais detalhes sobre o código Java do conector **SQLite-Redis Sentinel** ver o Anexo "SQLite-Redis Java Code Connector".

Configuração do ficheiro **redis.conf** da base de dados Redis (**Master**) para Windows 10.

Adicione as seguintes alterações ou directivas ao ficheiro, guarde as alterações e inicie o servidor Redis

Comece por encontrar a configuração **tcp-keepalive** e fixe-a em 60 segundos, como sugerido pelos comentários. Isto ajudará a Redis a detectar problemas de rede ou de serviço:

tcp-keepalive 60

Encontre a directiva de **passagem obrigatória** e configure-a com uma senha forte. Embora o seu tráfego Redis deva ser protegido de terceiros, isto fornece autenticação ao Redis. Uma vez que a Redis é rápida e não classifica as tentativas de frase limite, escolha uma frase limite forte e complexa para proteger contra as tentativas de força bruta:

requirepass type_your_network_master_password

exemplo:

Requerirpass FPqwedsLMdf76ass7asddfd2g45vBN8ty99

Finalmente, existem algumas configurações opcionais que poderá querer ajustar, dependendo do seu cenário de utilização.

Se não quiser que a Redis coloque automaticamente as chaves mais antigas e menos usadas à medida que se enche, pode desactivar a remoção automática da chave:

maxmemory-policy noevasion

Para garantias de durabilidade melhoradas, pode permitir a persistência de ficheiros adicionais apenas. Isto ajudará a minimizar a perda de dados no caso de uma falha do sistema à custa de ficheiros maiores e de um desempenho ligeiramente mais lento:

anexamente sim

appendfilename "redis-staging-ao.aof"

Proceder para guardar as alterações e reiniciar o serviço Redis para Windows 10, parar com as teclas Ctrl + C e executar novamente a linha de comando CMD do Windows:

C:\redis_redis_directory redis_server

No nosso exemplo, podemos ver que temos um nó (**escravo**) ligado.

```

:\\com\\redis> redis-server redis.conf
1:C 31 May 2020 23:44:56.633 # o000o000o000 Redis is starting o000o000o000
1:C 31 May 2020 23:44:56.634 # Redis version=5.0.7, bits=64, commit=00000000, modified=0,
id=51, just started
1:C 31 May 2020 23:44:56.634 # Configuration loaded
1:M 31 May 2020 23:44:56.635 * Increased maximum number of open files to 10032 (it was ori
ginally set to 1024).

Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 51

http://redis.io

1:M 31 May 2020 23:44:56.648 # WARNING: The TCP backlog setting of 511 cannot be enforced
because /proc/sys/net/core/somaxconn is set to the lower value of 128.51:May 31 May 2020 23:4
:56.648 # Server initialized
1:M 31 May 2020 23:44:56.649 # WARNING overcommit_memory is set to 0! Background save may
fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/s
ysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to t
ake effect.
1:M 31 May 2020 23:44:56.669 * DB loaded from append only file: 0.020 seconds
1:M 31 May 2020 23:44:56.669 * Ready to accept connections
1:M 31 May 2020 23:49:57.013 * 10 changes in 300 seconds. Saving...
1:M 31 May 2020 23:49:57.031 * Background saving started by pid 82
2:C 31 May 2020 23:49:57.052 * DB saved on disk
1:M 31 May 2020 23:49:57.133 * Background saving terminated with success
1:M 31 May 2020 23:50:24.600 * Replica 192.168.1.68:6379 asks for synchronization
1:M 31 May 2020 23:50:24.602 * Full resync requested by replica 192.168.1.68:6379
1:M 31 May 2020 23:50:24.602 * Starting BGSAVE for SYNC with target: disk
1:M 31 May 2020 23:50:24.619 * Background saving started by pid 83
3:C 31 May 2020 23:50:24.642 * DB saved on disk
1:M 31 May 2020 23:50:24.670 * Background saving terminated with success
1:M 31 May 2020 23:50:24.689 * Synchronization with replica 192.168.1.68:6379 succeeded

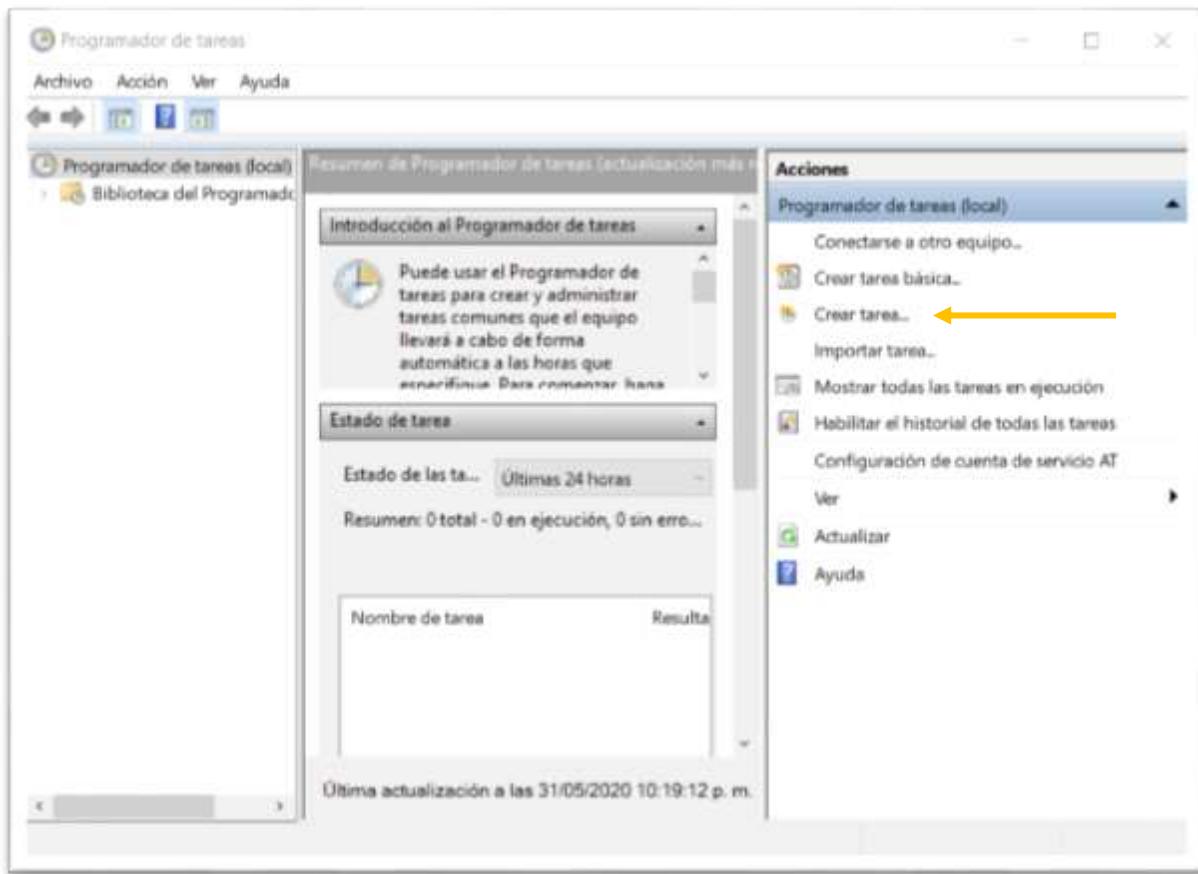
```

Podemos ver que sincronizamos um nó com o IP 192.168.1.68 na porta por defeito 6379.

Agora precisamos de agendar no sistema operativo Window 10 a tarefa de executar automaticamente o conector **SQLite-Redis Sentinel**. Fazemos isto com a ferramenta Windows 10, executamo-lo a partir da parte inferior esquerda, digitando "Programador de Tarefas".



Iremos criar uma nova tarefa "Criar tarefa" onde incluiremos a execução do conector.

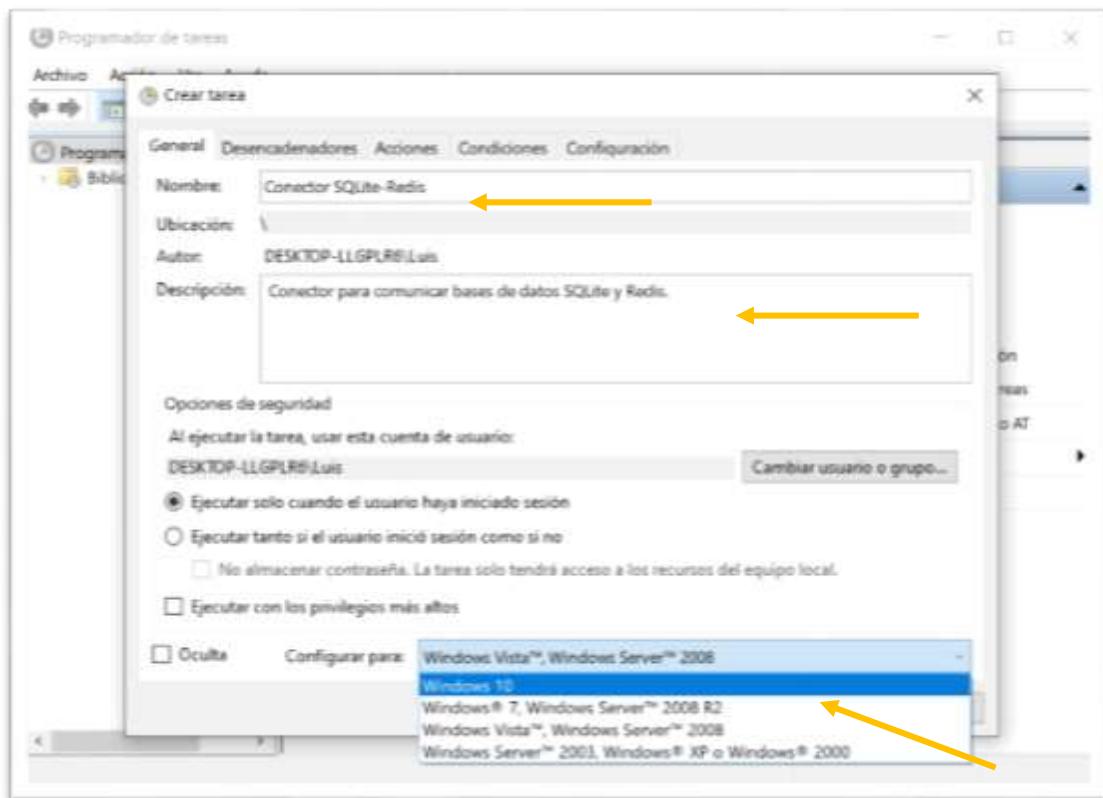


Clicando em "Criar tarefa" abrirá uma janela adicional onde temos de dar os seguintes parâmetros no separador "Geral":

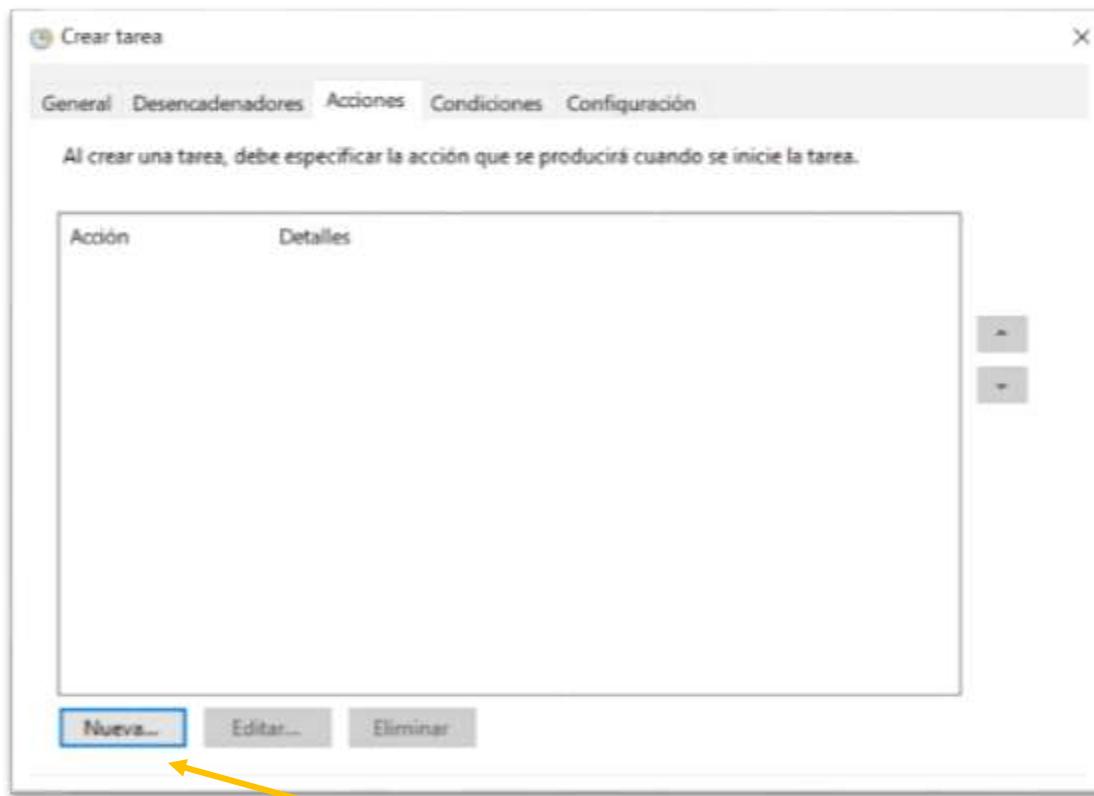
Nome: task_name

Descrição: opcional

Configurar para: select_operating_system_windows_version



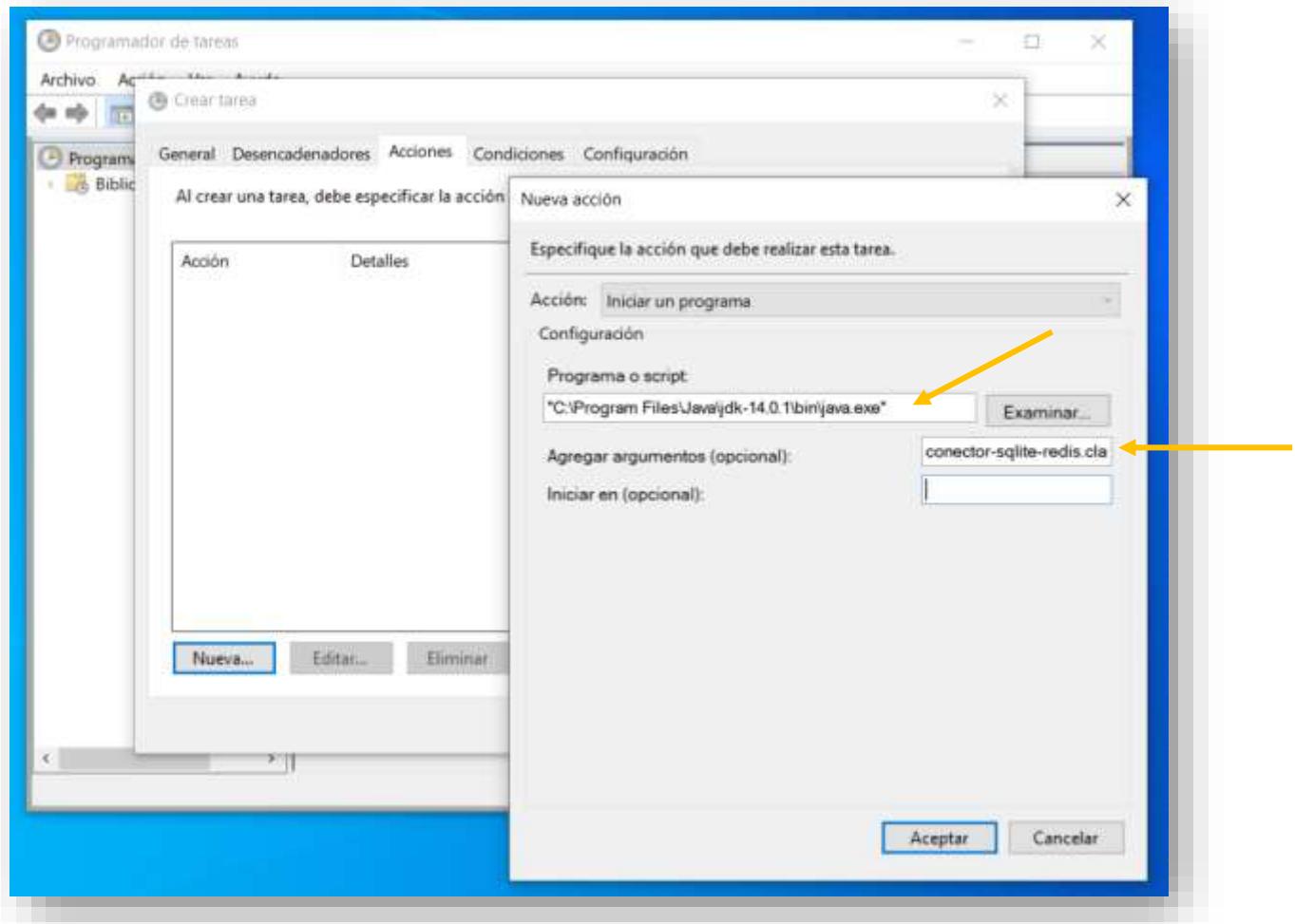
Alterar clicando no separador "Ações" e clicar no botão "Novo":



Damos os seguintes parâmetros:

Programa ou guião: conector_caminho

Acrecentar argumentos: nome do conector



Os parâmetros acima podem variar dependendo da localização do conector. A ideia principal é a execução do programa **conector-sqlite-redis-v1.class** como normalmente é feito na linha de comando:

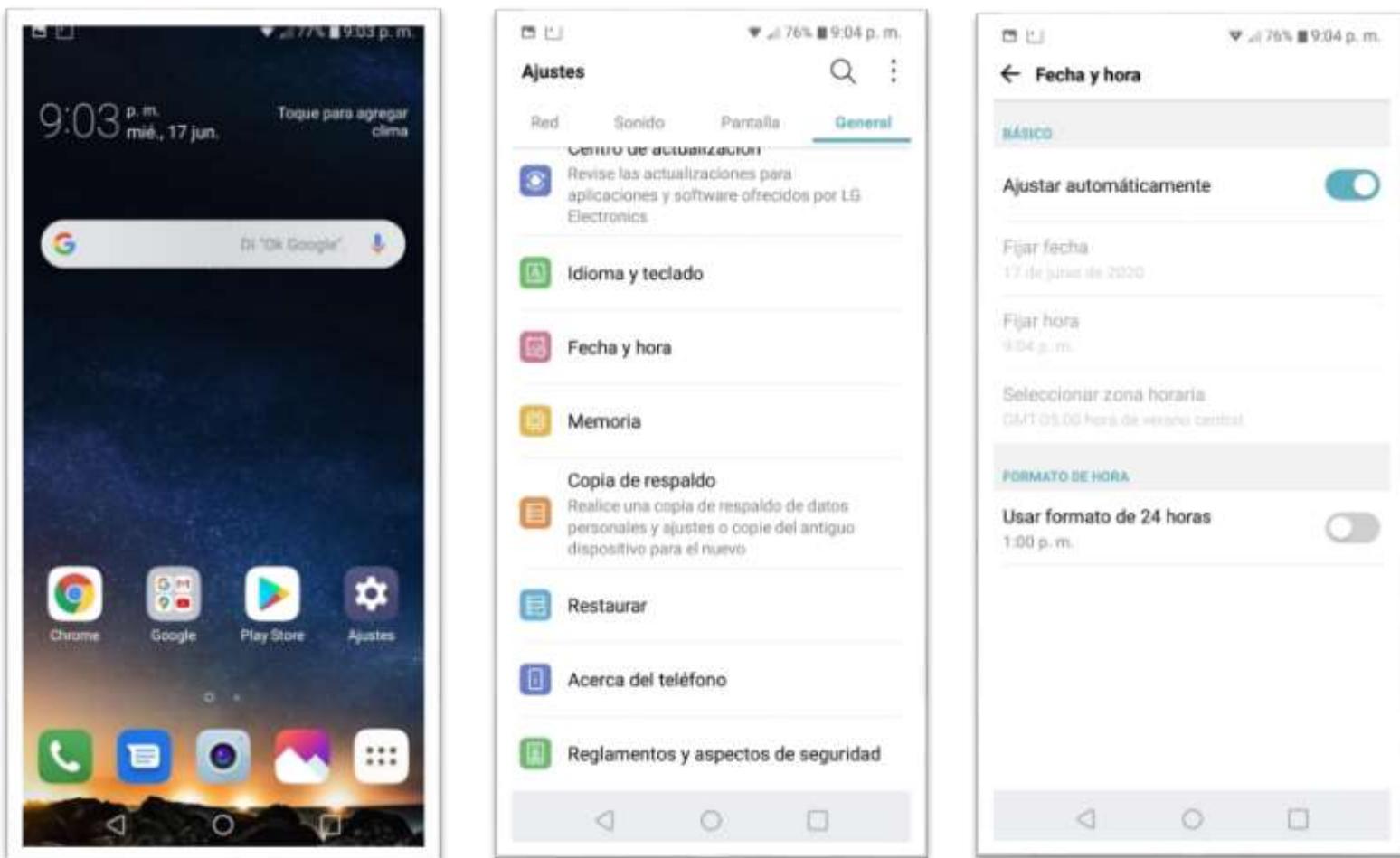
```
C:\jdk_directory java conector-sqlite-redis-v1
```

Para terminar devemos escolher o separador "Triggers", neste daremos os parâmetros de quando (dia, hora, minutos) queremos que a tarefa seja executada, estes parâmetros são baseados nas regras de negócio que o sistema Mini BlocklyChain terá de ser criado.

10. Sincronização em nós de sistema (telemóvel) minutos e segundos.

É muito importante que todos os nós estejam sincronizados principalmente na parte dos minutos e segundos. Uma vez que quando é feito o envio ou publicação num determinado momento da fila de transação todos os nós devem ser sincronizados, uma vez que isto dependerá do gestor de tarefas que será estabelecido no sistema local com a ferramenta CRON para ser executado ao mesmo tempo em todos os nós, isto fará com que todos os nós tenham a mesma probabilidade de poder ganhar o direito de ser o escolhido para poder processar a fila de transação e poder gerar o novo bloco para poder adicioná-lo à cadeia de blocos do sistema. Para sincronizar os nós, temos duas opções.

A primeira opção da sincronização do nó, conseguiremos fazê-lo de uma forma simples. No nosso dispositivo é através da opção interna que inclui o sistema Android, teremos de ir à parte de **Definições > Data e Hora > Ajustar automaticamente**



Com a configuração anterior teremos sincronizado em minutos e segundos todos os nós do sistema, independentemente do país do mundo já sincronizado, com base em cada área geográfica, possivelmente o que muda é o tempo mas, dependendo dos minutos e segundos, isto é suficiente para que possamos executar o processo de tarefas numa base programada com a ferramenta cron em todos os nós para executar cada vez em minutos, ou seja, podemos criar uma tarefa em crontab para executar a cada 10 minutos ou 30 minutos, dependendo do desenho de cada sistema.

Isto será útil quando se utilizar a rede de comunicação "Peer to Peer", no caso de se utilizar a rede de backup, este processo não se aplicará, uma vez que a distribuição da fila de transacções é feita num modelo cliente-servidor e o servidor é aquele que controla a ferramenta cron.

Referência: <https://appinventor.mit.edu/explore/blogs/karen/2016/08.html>

A segunda opção é utilizar um API externo onde executaremos o comando Curl através da extensão (**ConnectorSSHClient**).

O local onde iremos utilizar os serviços externos do NTP (Network Time Protocol) é:

<http://worldtimeapi.org/>

Agora vamos procurar uma forma de obter o tempo dos servidores NTP que estão localizados em todo o mundo e que nos ajudará a conseguir que todos os nós tenham uma consulta a uma determinada hora na mesma data e hora.

Exemplo de uma consulta com extensão (**ConnectorSSHClient**).

\$ encaracolar "http://worldtimeapi.org/api/timezone/America/Mexico_City"

Fizemos a ligação ao terminal Termux:



Executamos o comando Curl:



Devemos ter em conta que o resultado do comando Curl será no formato JSON, algo semelhante ao resultado seguinte:

```

abbreviation: CDT
client_ip: "200.77.16.151"
datetime: "2020-06-18T14:16:57.750466-05:00"
day_of_week: 4
day_of_year: 170
dst: true
dst_from: "2020-04-05T08:00:00+00:00"
dst_offset: 3600
dst_until: "2020-10-25T07:00:00+00:00"
raw_offset: -21600
timezone: "America/Mexico_City"
unixtime: 1592507817
utc_datetime: "2020-06-18T19:16:57.750466+00:00"
utc_offset: "-05:00"
week_number: 25

```

Também o resultado poderia ser em formato JSON sem os dados formatados em ou JSON em forma linear, como se mostra abaixo:

```
{"abreviatura": "CDT", "client_ip": "200.77.16.151", "datetime": "2020-06-18T14:19:07.216800-05:00", "dia_da_semana": 4, "dia_do_ano": 170, "dst": true, "dst_de": "2020-04-05T08:00:00+00:00", "dst_offset": 3600, "dst_até": "2020-10-25T07:00:00+00:00", "raw_offset": -21600, "timezone": "America/Mexico_City", "unixtime": 1592507947, "utc_datetime": "2020-06-18T19:19:07.216800+00:00", "utc_offset": "-05:00", "week_number": 25}
```

Qualquer das duas formas anteriores teremos de filtrar a informação através de uma extensão JSON existente como "JSONTOOLS" ou utilizar filtros no App Inventor no processamento de texto para obter apenas a hora, dia ou segundos, de acordo com a necessidade de cada sistema. Após o processamento do resultado, pode ser feita uma comparação lógica e com base nessa comparação podemos executar uma tarefa já programada com o serviço "cron" que mais tarde veremos a sua configuração em cada nó.

Referência de extensão do JSONTOOLS:

<https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html>

Revimos agora duas opções para sincronizar a hora dos nós. Vamos continuar a configurar o serviço "cron" em cada nó.

Configuração de agendamento automático de tarefas para execução com o serviço **CRON** em sistemas Android (nós do sistema).

Primeiro temos de compreender como funciona o programador automático.

O serviço cron normalmente todos os sistemas têm este pré-instalado e onde todas as tarefas a serem executadas automaticamente estão agendadas num ficheiro chamado crontab.

Editamos o ficheiro crontab com **\$ crontab -e**, utilizamos o editor **vi**, no interior utilizamos o formato:

```
# m h dom mon dow comando de utilizador
```

onde:

- **m** corresponde ao minuto em que o guião será executado, o valor vai de 0 a 59
- **h a** hora exacta, o formato é 24 horas, os valores vão de 0 a 23, sendo 0 12:00 meia-noite.
- **dom** refere-se ao dia do mês, por exemplo, pode especificar 15 se quiser correr a cada 15
- **dow** significa o dia da semana, pode ser numérico (0 a 7, onde 0 e 7 são domingo) ou as 3 primeiras letras do dia em inglês: mon, tue, wed, thu, fri, sat, sun.
- **utilizador** define o utilizador que irá executar o comando, pode ser root, ou um utilizador diferente desde que tenha permissões para executar o script.
- refere-se ao comando ou caminho absoluto do script a ser executado, exemplo: /home/user/scripts/update.sh, se chamar um script deve ser executável

Para tornar claro alguns exemplos de tarefas cron explicados:

```
15 10 * * * utilizador /home/utilizador/scripts/update.sh
```

Executará o update.sh todos os dias às 10:15 da manhã

```
15 22 * * * utilizador /home/utilizador/scripts/update.sh
```

Executará o update.sh todos os dias às 22:15 p.m.

```
00 10 * * 0 root apt-get - e actualizar a raiz do utilizador
```

Fará uma actualização todos os domingos às 10:00 da manhã.

```
45 10 * * raiz do sol apt-get - e actualizar
```

O utilizador root executará uma actualização todos os domingos (sol) às 10:45 da manhã.

Guardámos as alterações no editor e com isto terminámos a configuração do serviço cron. No caso de não ter o cron instalado no sistema, pode fazê-lo com o seguinte comando:

```
$ apt install cron
```

2. Instalação e configuração de nós de rede - Telemóveis.

Comecemos pela rede de comunicações para os nós que irão utilizar o Mini BlocklyChain.

Primeiro precisamos de um ambiente Linux uma vez que cada sistema Android é baseado em Linux para segurança e flexibilidade em ferramentas, utilizaremos o terminal "Termux" que contém aquele ambiente onde instalaremos a rede de comunicações.

Termux é um emulador Linux onde iremos instalar os pacotes necessários para criar a nossa rede de comunicação entre nós.

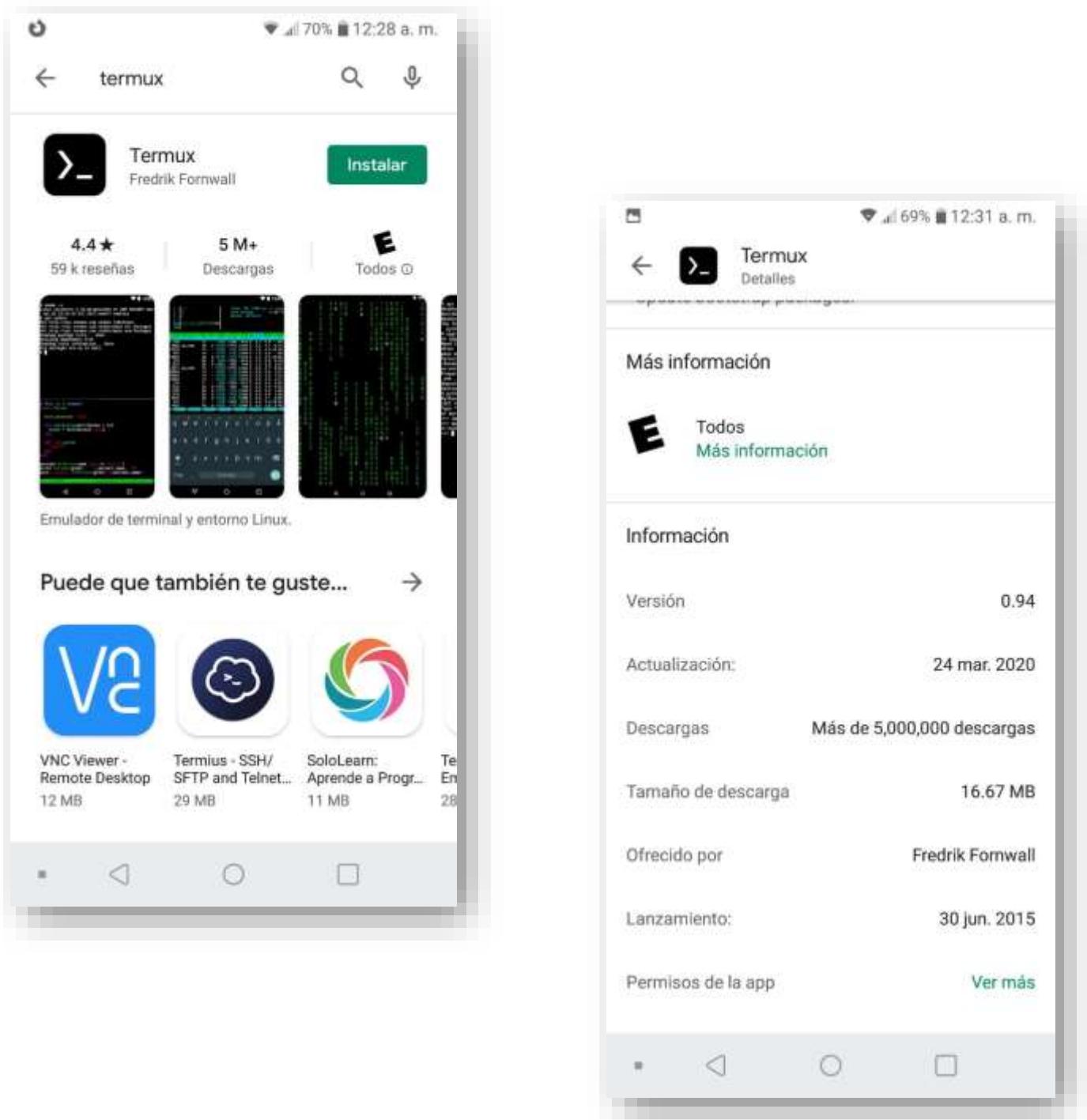
Uma das principais vantagens de utilizar Termux é que pode instalar programas sem ter de "rodar" o telemóvel (Smartphone), o que garante que não se perde a garantia do fabricante devido a esta instalação.

Instalação de Termux.

A partir do seu telemóvel, vá para a aplicação ícone do Google Play (play.google.com).



Pesquisar por aplicação "Termux", seleccioná-la e iniciar o processo de instalação.



Início da aplicação Termux.

Depois de começarmos, teremos de executar os dois comandos seguintes para efectuar actualizações do emulador do sistema operativo Linux:

\$ apt update

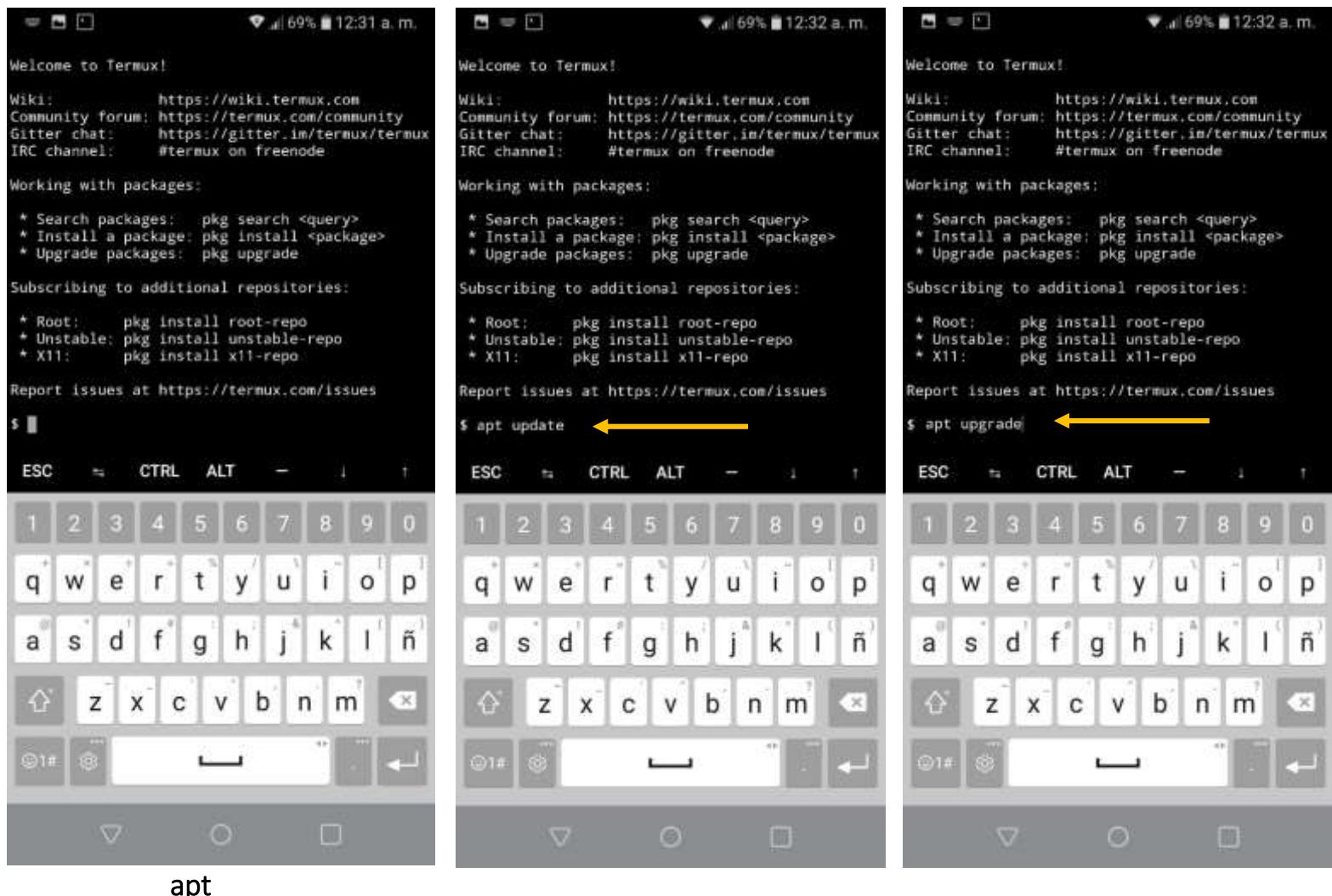
\$ apt upgrade

Confirmar todas as opções Y(Sim)...

Termux

Home \$ actualização

do apt \$ actualização do



11. Configuração de armazenamento dentro da Termux.

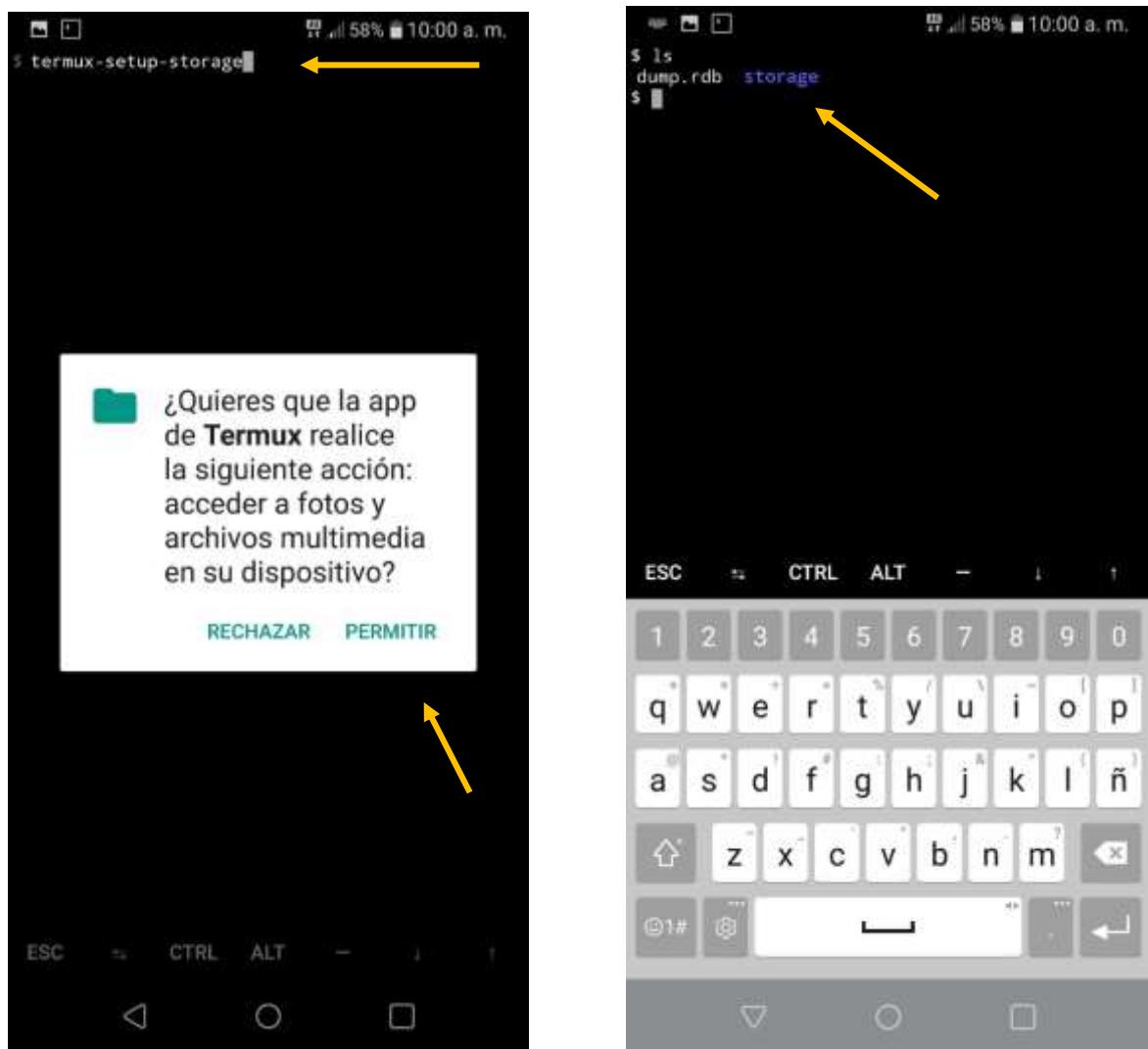
Após ter actualizado e actualizado o sistema Termux, começaremos a configurar a forma de visualizar o armazenamento interno do telefone no sistema Termux, o que o ajudará a poder trocar informações entre o Termux e as nossas informações no telefone.

Isto pode ser feito de forma simples e rápida, executando o seguinte comando num terminal Termux.

Termux-setup-storage \$

Quando executa o comando anterior, aparece uma janela a pedir-lhe para confirmar a criação de um **armazenamento** virtual (directório) no Termux. Verificamos ao dar o comando:

\$ ls



12. Instalação de rede "Tor" e instalação de "Syncthing".

\$ apt install tor

\$ apt install syncthing

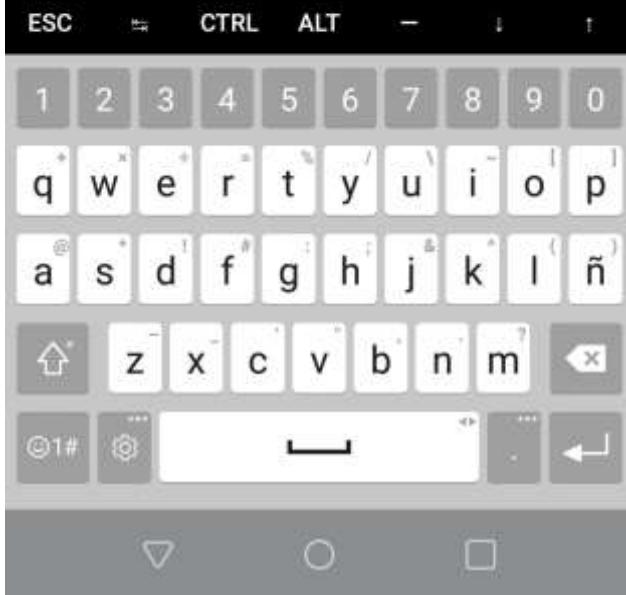
Aceitar instalação digitando Y maiúsculo em ambos os casos, se solicitado.

\$ apt install tor

```
$ apt install tor
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
libevent
The following NEW packages will be installed:
libevent
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 2319 kB of archives.
After this operation, 12.6 MB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
```

\$ apt install syncthing

```
$ apt install syncthing
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
syncthing
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 6407 kB of archives.
After this operation, 19.3 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ages-24 stable/main arm syncthing arm 1.5.0 [6407
kB]
27% [1 syncthing 2193 kB/6407 kB 34%]
```



13. Instalação da base de dados "Redis" e do servidor SSH (Secure Shell).

\$ apt install redis

\$ apt install openssh

\$ apt install sshpass

\$ apt install redis

```
$ apt install redis
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  redis
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 528 kB of archives.
After this operation, 3056 kB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm redis arm 6.0.1 [528 kB]
Fetched 528 kB in 1s (296 kB/s)
Selecting previously unselected package redis.
(Reading database ... 3265 files and directories
currently installed.)
Preparing to unpack .../archives/redis_6.0.1_arm
.deb ...
Unpacking redis (6.0.1) ...
Setting up redis (6.0.1) ...
$
```



\$ apt install openssh

```
$ apt install openssh
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be instal
led:
  krb5-libs libdb5 libedit termux-auth
The following NEW packages will be installed:
  krb5-libs libdb5 libedit openssh termux-auth
0 upgraded, 6 newly installed, 0 to remove and 0
not upgraded.
Need to get 2255 kB of archives.
After this operation, 11.9 MB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm libdb5 arm 18.1.32-4 [465
kB]
Get:2 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm krb5-libs arm 1.18.1 [839 kB]
24% [2 krb5-libs 131 kB/839 kB 16%]
```



\$ apt install sshpass

```
$ apt install sshpass
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  sshpass
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 7158 B of archives.
After this operation, 57.3 kB of additional disk
space will be used.
0% [Working]
```



Terminámos com a instalação da rede de comunicação, continuamos com a configuração dos pacotes: Tor, Syncthing e Redis DB.

14. Configuração do servidor SSH no telemóvel (smartphone).

Permitiremos que o servidor SSH no telemóvel seja capaz de se ligar do nosso PC ao telemóvel e de trabalhar de uma forma mais rápida e confortável, servirá também para

verificar se o serviço do servidor SSH no telemóvel funciona correctamente, uma vez que o utilizaremos na rede de comunicação no Mini BlocklyChain.

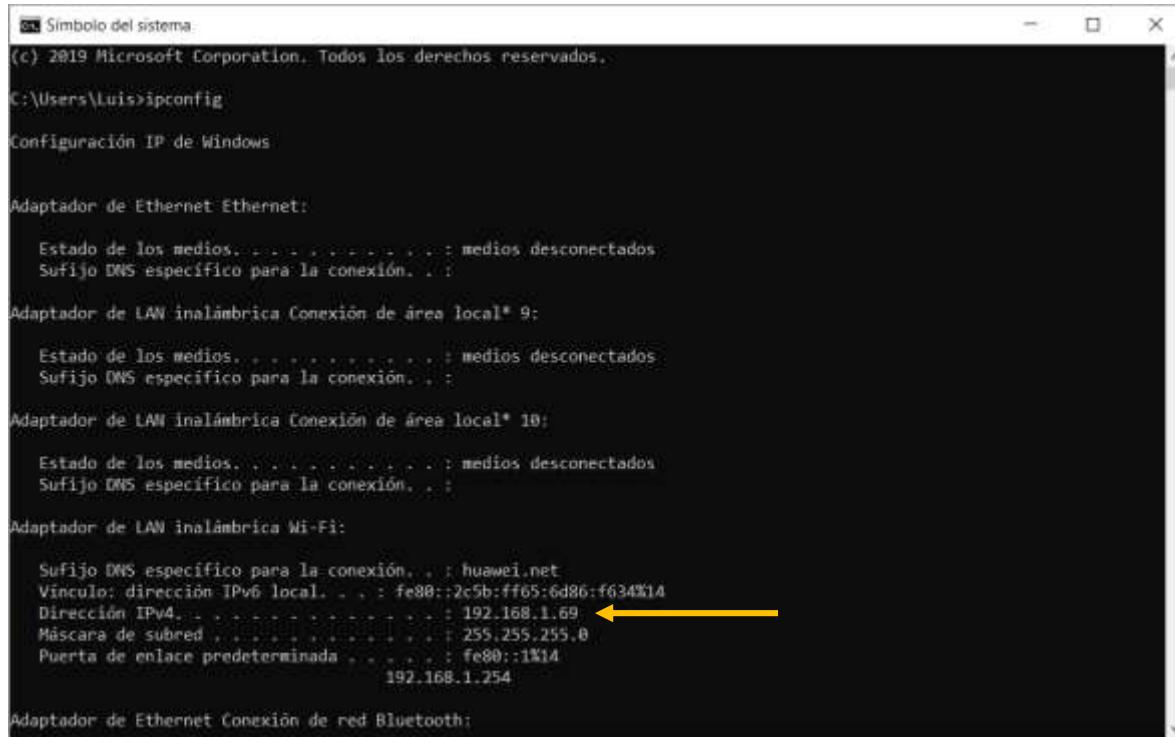
A primeira coisa que temos de fazer é ligar o telemóvel e o PC à **mesma rede WiFi** para que eles se possam ver uns aos outros. Os IPs ou endereços devem ser semelhantes a 192.168.XXX.XXX os valores XXX são números variáveis que são atribuídos aleatoriamente em cada computador.

Este exemplo foi testado num telemóvel LG Q6 e num PC com Windows 10 Home.

Verificar o IP ou endereço que o PC tem ligado ao WiFi temos de abrir um terminal no Windows.

No painel inferior onde a lupa de pesquisa é escrita cmd e premir a tecla Enter. Um terminal será aberto e nele escreveremos o comando:

C:\Nome_do_utilizador> ipconfig



```
Símbolo del sistema
(c) 2019 Microsoft Corporation. Todos los derechos reservados.
C:\Users\Luis>ipconfig

Configuración IP de Windows

Adaptador de Ethernet:
Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . . . : 

Adaptador de LAN inalámbrica Conexión de área local* 9:
Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . . . : 

Adaptador de LAN inalámbrica Conexión de área local* 10:
Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . . . : 

Adaptador de LAN inalámbrica Wi-Fi:
Sufijo DNS específico para la conexión. . . . . : huawei.net
Vínculo: dirección IPv6 local. . . . . : fe80::2c5b:ff65:6d86:f634%14
Dirección IPv4. . . . . : 192.168.1.69
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . : fe80::1%14
192.168.1.254

Adaptador de Ethernet Conexión de red Bluetooth:
```

Mostrar-nos-á o IP atribuído ao PC neste é 192.168.1.69 mas o mais provável é que seja diferente em cada caso.

NOTA: O endereço onde diz "endereço IPv4" deve ser tomado, não deve ser confundido com o Gateway.

Agora, no caso do telemóvel no terminal Termux devemos digitar o seguinte comando para saber o nome do nosso utilizador que vamos utilizar para nos ligarmos ao servidor SSH que tem o nosso telefone, executamos o seguinte comando:

\$ whoami

Mais tarde temos de dar uma palavra-chave a este utilizador, pelo que temos de executar o seguinte comando:

Senha de \$

Irá pedir-nos para escrever uma palavra-passe e premir Enter, mais uma vez pede-nos a palavra-passe que confirmámos e premir Enter, se tiver sido "**Nova palavra-passe foi definida com sucesso**" em caso de marcação de um erro é possível que a palavra-passe não tenha sido digitada correctamente. Realizar o procedimento de novo.

E depois, para saber que IP temos no Termux digitamos o seguinte comando, o IP está após a palavra "inet":

\$ ifconfig -a

The image consists of two screenshots of the Termux application running on an Android phone. The top screenshot shows a terminal session where the user has run the command `$ whoami`, which returns `u0_a263`. The user then runs `$ passwd` and is prompted to enter a new password. They type a password and then retype it. A message indicates that the password was successfully set. The bottom screenshot shows the result of running `$ ifconfig -a`. It displays network interface information for `wlan0` and `eth0`. The `wlan0` interface is shown with its IP configuration: `inet 192.168.1.68 netmask 255.255.255.0 broadcast 192.168.1.255`. A yellow arrow points to this line. The `eth0` interface is also listed with its statistics.

```

$ whoami
u0_a263
$ passwd
New password:
Retype new password:
New password was successfully set.
$ 

$ ifconfig -a
eth0      Link encap:Ethernet HWaddr 00:0C:29:1A:0D:02
          brd 00:0C:29:1A:0D:02  MTU:1500  RX:0 errors:0 dropped:0 overruns:0 carrier:0 collisions:0
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
          Interrupt:15

wlan0     Link encap:IPv4/BROADCAST/MULTICAST  HWaddr 00:0C:29:1A:0D:01
          brd 192.168.1.255  MTU:1500  RX:0 errors:0 dropped:0 overruns:0 carrier:0 collisions:0
          inet 192.168.1.68  netmask 255.255.255.0  broadcast 192.168.1.255
                  inet6 fe80::c1ff:fe00:1a0d%wlan0  brd fe80::ff:fe00:1a0d%wlan0  scopeid -1
                          link-layer
          RX bytes:908745  bytes:947916536 (904.0 MiB)
          RX errors:0  dropped:0  overruns:0  frame:0
          TX bytes:601034  bytes:93496881 (89.1 MiB)
          TX errors:0  dropped:0  overruns:0  carrier:0  collisions:0
$ 

```

Agora é altura de iniciar o serviço de servidor SSH no seu telefone para que possa receber sessões do seu PC. Executamos o seguinte comando no terminal Termux, este comando não dá qualquer resultado.

\$ sshd



Agora teremos de instalar um programa no PC que irá comunicar com o servidor SSH do telefone a partir do PC.

Temos de ir a <https://www.putty.org>

Seleccione onde está o link "Pode descarregar PuTTY aqui".



Download PuTTY

PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the with source code and is developed and supported by a group of volunteers.

You can download PuTTY [here](#).



Below suggestions are independent of the authors of PuTTY. They are not to be seen as



Bitvise SSH Client

Bitvise SSH Client is an SSH and SFTP client for Windows. It is developed and supported prof supports all features supported by PuTTY, as well as the following:

- graphical SFTP file transfer;
- single-click Remote Desktop tunnelling;
- auto-reconnecting capability;
- dynamic port forwarding through an integrated proxy;
- an FTP-to-SFTP protocol bridge.

Bitvise SSH Client is **free to use**. You can [download it here](#).

Escolha a versão de 32 bits, não importa se o seu sistema é de 64 bits, funcionará bem.

Download PuTTY: latest releases

[Home](#) | [FAQ](#) | [Feedback](#) | [Licence](#) | [Updates](#) | [Mirrors](#)
Download: **Stable** · [Snapshot](#) | [Docs](#) | [Changelog](#)

This page contains download links for the latest released version of PuTTY. Currently this is 0.73, released on 2019-09-29.

When new releases come out, this page will update to contain the latest, so this is a good page to bookmark or link to. Alternative Release versions of PuTTY are versions we think are reasonably likely to work well. However, they are often not the most up-to-date, so check the [development snapshots](#), to see if the problem has already been fixed in those versions.

Package files

You probably want one of these. They include versions of all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

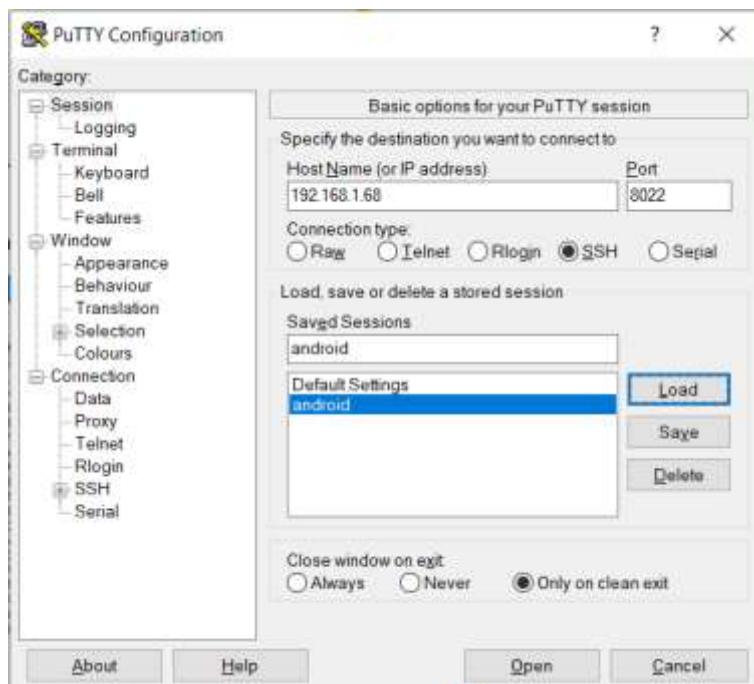
MSI ("Windows Installer")

32-bit:	putty-0.73-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.73-installer.msi	(or by FTP)	(signature)

Unix source archive

.tar.gz:	putty-0.73.tar.gz	(or by FTP)	(signature)
----------	-----------------------------------	------------------------------	-------------------------------

Uma vez descarregado para o seu PC, execute-o e instale-o com as opções predefinidas. Em seguida, iniciar a aplicação PuTTY.



Nesta sessão iremos introduzir os dados do nosso servidor Openssh que instalámos no telemóvel.

Introduza o IP do telemóvel.

HostName ou endereço IP:

192.168.1.68 (exemplo IP)

Porto:

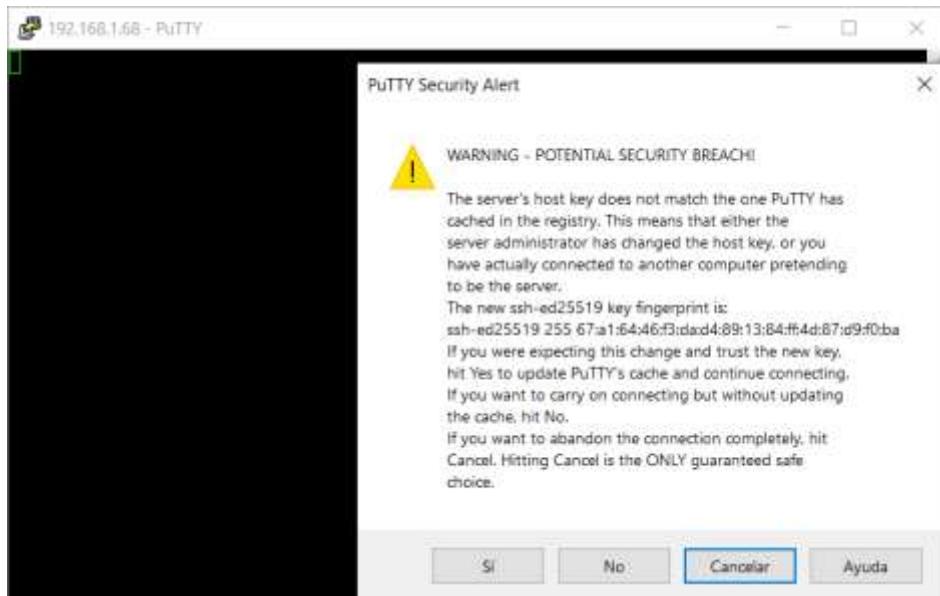
8022 (Porta predefinida do servidor SSH móvel).

Podemos dar um nome à

sessão em "Sessões Guardadas" e clicar no botão Guardar.

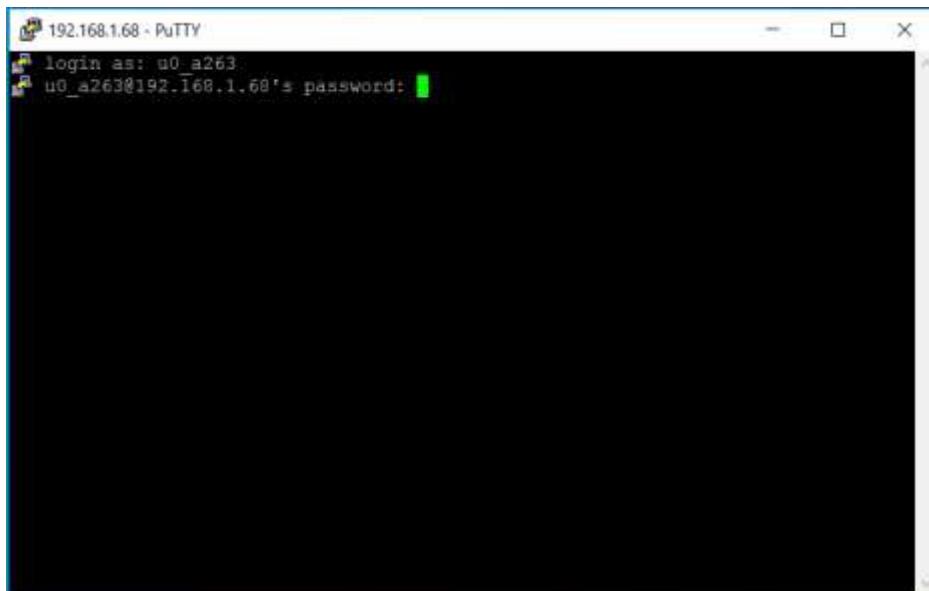
Mais tarde, na parte inferior, premimos para abrir uma ligação ao servidor dando o botão "Open".

No PC, quando se ligar pela primeira vez, ser-lhe-á pedida a confirmação da chave de encriptação da informação, clicando no botão "Sim".

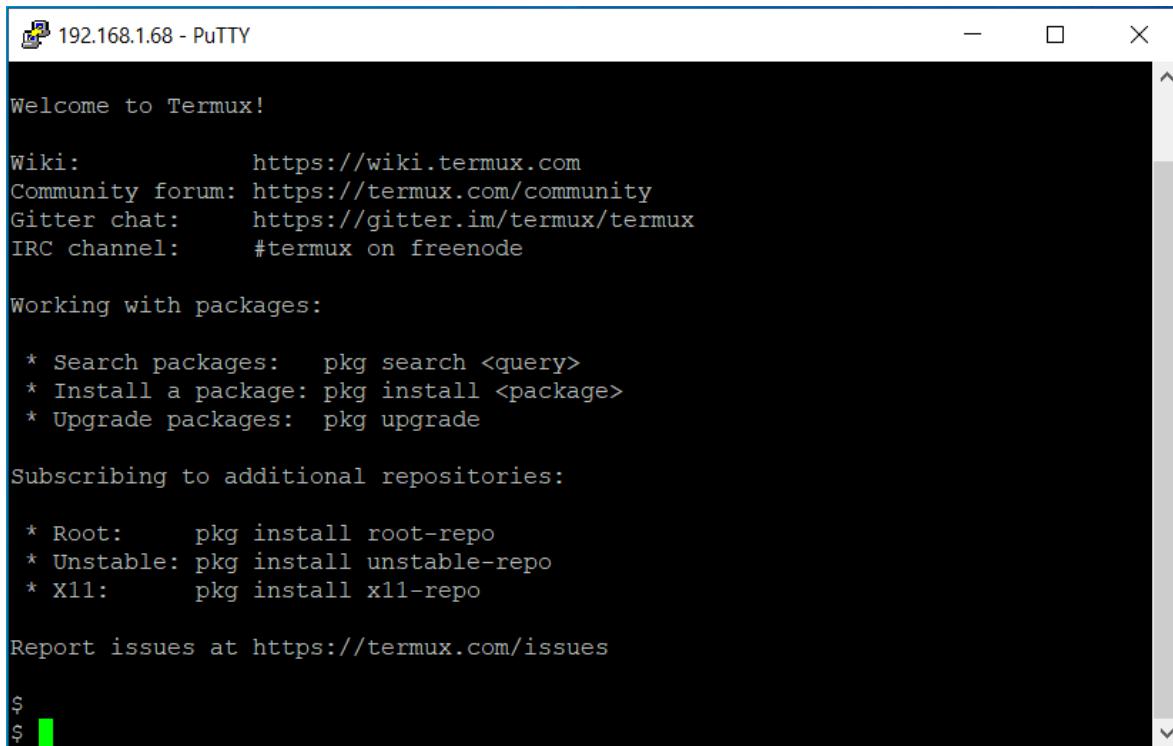


Mais tarde, ser-nos-á pedido o utilizador com o qual nos vamos ligar. Utilizaremos a informação que obtivemos anteriormente (utilizador e palavra-chave).

No **Login como:** temos de introduzir o nosso utilizador e dar Enter, depois pediremos novamente a palavra-passe e daremos o botão Enter.



Se os dados estiverem correctos, estaremos numa sessão SSH (Secure Shell) realizada a partir do PC (Cliente) ao telefone (Servidor SSH).



```
Welcome to Termux!

Wiki: https://wiki.termux.com
Community forum: https://termux.com/community
Gitter chat: https://gitter.im/termux/termux
IRC channel: #termux on freenode

Working with packages:

* Search packages: pkg search <query>
* Install a package: pkg install <package>
* Upgrade packages: pkg upgrade

Subscribing to additional repositories:

* Root: pkg install root-repo
* Unstable: pkg install unstable-repo
* X11: pkg install x11-repo

Report issues at https://termux.com/issues

$
```

NOTA IMPORTANTE: Lembre-se que o IP (endereço) do PC e o IP (endereço) do telemóvel ligado ao mesmo WiFi irá provavelmente mudar cada vez que desligarmos e voltarmos a ligar, pelo que temos de verificar duas vezes quais os endereços de cada dispositivo, o que garantirá o sucesso da ligação entre os dispositivos através do servidor SSH do telefone e o PC (Cliente).

Até agora só nos conseguimos ligar à mesma rede WiFi, mas se movermos o nosso telefone para fora da mesma rede que o PC, não conseguiremos ligar porque existem diferentes redes envolvidas em passar por outros dispositivos de comunicação mais complicados. Resolveremos isto quando criarmos a rede "Tor".

Lembre-se que esta ligação é feita apenas para verificar o serviço do servidor que instalámos no telefone e para ter um ambiente de trabalho mais confortável com uma sessão remota de um PC para o telefone.

15. Configuração de rede "Tor" com serviço SSH (Secure Shell).

Com a sessão remota a partir do PC, vamos começar a configurar a rede "Tor" com o serviço SSH activado.

A importância de ter a rede "Tor" é dar aos dispositivos a propriedade de poder comunicar em qualquer parte do mundo através da Internet sem estar na mesma rede WiFi, não importa onde estejamos, poderemos ligar-nos e formar a rede "Peer to Peer" entre nós que é uma funcionalidade essencial da arquitectura do Mini BlocklyChain.

A rede "Tor" tem muita flexibilidade na sua configuração uma vez que tem vários parâmetros no seu ficheiro de configuração "torrc" este ficheiro está no caminho (`$_PREFIX/etc/tor/torrc`) no nosso caso com a sessão Termux do nosso PC saberemos digitando o seguinte comando:

```
$ echo $PREFIX
```

```
/dados/dados/com.termux/arquivos/usr
```

Por conseguinte, o ficheiro que precisamos de editar estará no caminho:

```
PREFIX/etc/tor/torrc que é igual a /dados/dados/com.termux/files/usr/etc/tor/torrc
```

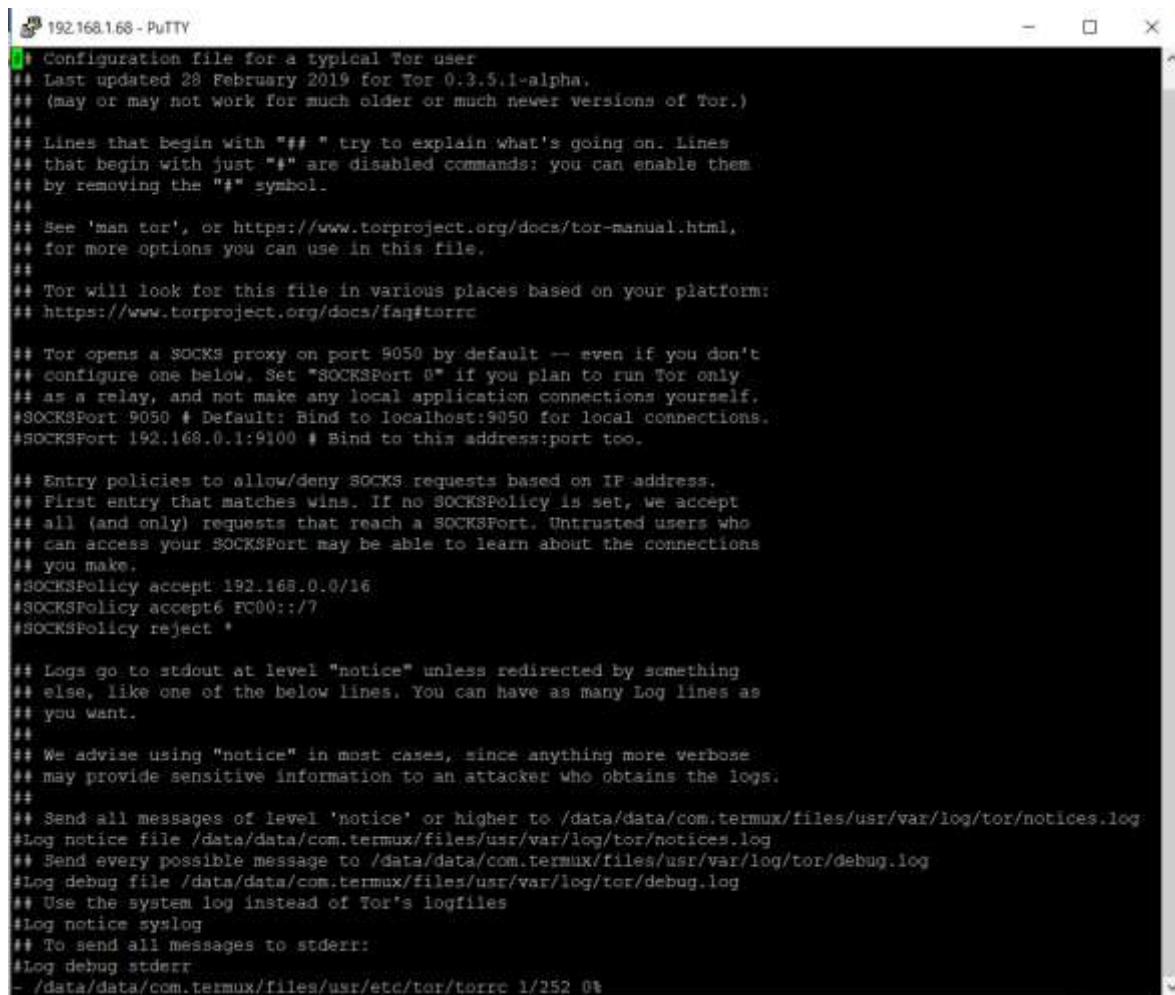
Procedemos à edição do ficheiro de configuração utilizando o editor de linha de comando "vi", executando o seguinte comando:

```
vi /dados/dados/com.termux/files/usr/etc/torrc
```

Ele irá editar esse ficheiro para nós, como exemplo:



Ficheiro "torrc" editado:



```

# Configuration file for a typical Tor user
## Last updated 28 February 2019 for Tor 0.3.5.1-alpha,
## (may or may not work for much older or much newer versions of Tor.)
##
## Lines that begin with "## ", try to explain what's going on. Lines
## that begin with just "#" are disabled commands: you can enable them
## by removing the "#" symbol.
##
## See 'man tor', or https://www.torproject.org/docs/tor-manual.html,
## for more options you can use in this file.
##
## Tor will look for this file in various places based on your platform:
## https://www.torproject.org/docs/faq#torrc

## Tor opens a SOCKS proxy on port 9050 by default -- even if you don't
## configure one below. Set "SOCKSPort 0" if you plan to run Tor only
## as a relay, and not make any local application connections yourself.
#SOCKSPort 9050 # Default: Bind to localhost:9050 for local connections.
#SOCKSPort 192.168.0.1:9100 # Bind to this address:port too.

## Entry policies to allow/deny SOCKS requests based on IP address.
## First entry that matches wins. If no SOCKSPolicy is set, we accept
## all (and only) requests that reach a SOCKSPort. Untrusted users who
## can access your SOCKSPort may be able to learn about the connections
## you make.
#SOCKSPolicy accept 192.168.0.0/16
#SOCKSPolicy accept6 FC00::/7
#SOCKSPolicy reject *

## Logs go to stdout at level "notice" unless redirected by something
## else, like one of the below lines. You can have as many log lines as
## you want.
##
## We advise using "notice" in most cases, since anything more verbose
## may provide sensitive information to an attacker who obtains the logs.
##
## Send all messages of level 'notice' or higher to /data/data/com.termux/files/usr/var/log/tor/notices.log
#Log notice file /data/data/com.termux/files/usr/var/log/tor/notices.log
## Send every possible message to /data/data/com.termux/files/usr/var/log/tor/debug.log
#Log debug file /data/data/com.termux/files/usr/var/log/tor/debug.log
## Use the system log instead of Tor's logfiles
#Log notice syslog
## To send all messages to stderr:
#Log debug stderr
- /data/data/com.termux/files/usr/etc/tor/torrc 1/252 0t

```

Neste ficheiro "torrc" teremos de adicionar ou utilizar as linhas que o ficheiro tem, fazendo as seguintes alterações, três linhas que são as seguintes

Sintaxe: SOCKSPort < número da porta de aplicação>

Exemplo: SOCKSPort 9050

A variável **SOCKSPort** diz-nos que esta tomada de comunicação sobre o protocolo TCP-IP utilizará o dispositivo móvel (telefone) por defeito e a porta 9050 será aberta ou em uso.

Sintaxe: HiddenServiceDir <Directório onde será guardada a configuração da aplicação>

Exemplo: HiddenServiceDir /data/data/com.termux/files/

A variável **HiddenServiceDir** diz-nos que este será o directório onde será armazenada a configuração do serviço que será utilizado através da rede Tor. Neste directório encontrará o ficheiro de configuração e segurança para o serviço, e neste directório encontrará um ficheiro chamado **hostname**.

Podemos ver o endereço atribuído pela rede Tor para o serviço específico criado no nosso caso está a criar um serviço SSH que será implementado na rede Tor, o directório que estamos a nomear como **hidden_ssh** conterá o ficheiro do **hostname**. Este directório não precisa de ser criado, porque quando iniciarmos o serviço de rede Tor ele será criado automaticamente.

Para ver o endereço fornecido pela rede Tor, podemos usar o comando "mais". Antes de usarmos este comando, devemos terminar de configurar o ficheiro "torrc" e registar o serviço de rede Tor de modo a que o **directório_sh oculto** e os ficheiros dentro dele sejam criados, como é o caso do ficheiro **hostname**.

mais /dados/dados/com.termux/arquivos/

O comando acima resultará num endereço com uma cadeia alfanumérica com uma extensão . cebola semelhante a :

uqwthf6ojdmoybyzvudoq3x2weq7k7rjitblhba3pjbawk2nvjmx3wer.onion

Finalmente, precisamos de adicionar a variável **HiddenServicePort** ao ficheiro de configuração "torrc". Este parâmetro diz à rede Tor qual a porta que a aplicação para a qual nos estamos a inscrever irá utilizar através da rede Tor.

Sintaxe: HiddenServicePort <Port de partida>>IP local ou público>: <Port de partida>

O

HiddenServicePort <Port de partida>

Exemplos:

HiddenServicePort 22 127.0.0.1:8022

O

HiddenServicePort 8022

Com o acima exposto, terminámos a configuração da rede Tor para serviços genéricos e serviço SSH (Secure Shell). Este serviço será utilizado para a comunicação de actualização de bases de dados SQLite, onde estaremos a guardar os processos de validação do nosso sistema Mini BlocklyChain.

Continuamos com a configuração da rede de comunicações Mini BlocklyChain.

16. Configuração do sistema Peer to Peer com sincronização manual.

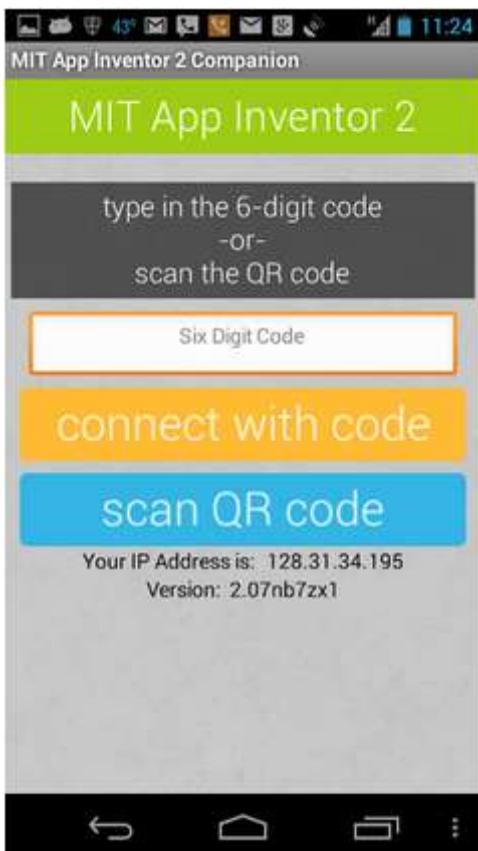
Uma arquitectura "Peer to Peer" é fundamental num sistema de tecnologia de cadeia de bloqueio porque esta arquitectura dá dois pontos centrais nos processos de comunicação do sistema, um é fornecer a mesma informação actualizada (sincronizada) em qualquer altura em todos os nós, independentemente de os nós estarem numa rede privada (Wifi) ou numa rede pública (Internet) ou num híbrido destes dois e um segundo ponto deste tipo de arquitectura é que eles não dependem de um intermediário (servidor) para transferir, actualizar ou consultar informação entre nós. Tudo isto se deve ao facto de a comunicação ser feita directamente entre os nós, no nosso caso Mini BlocklyChain a comunicação é feita directamente entre os telefones que formam a rede sem um intermediário.

Para esta tarefa utilizaremos a ferramenta Syncthing de código aberto que funciona com base numa arquitectura "Peer to Peer".

A configuração do Syncthing para dispositivos (telemóveis) será vista manual e automaticamente. O formulário manual é aplicado em sincronização com até 5 nós, no caso de ter um grande número de configuração automática é óptimo, o processo centra-se no registo dos nós com os quais queremos realizar a sincronização da informação seleccionada.

Os requisitos para começar são:

1. Iniciaram o serviço da rede Tor.
2. (opcional - recomendado) Ter instalado uma aplicação que possa ler códigos QR, sugerimos a aplicação Android inventora da aplicação que é fácil e simples de instalar a partir do Google Play e mais tarde neste manual iremos utilizá-la na secção "Definição e utilização de blocos em Mini Blocklychain".
3. Ter iniciado o serviço de Synthing.



Mostramos a aplicação App Inventor que utilizaremos para a leitura de códigos QR que nos servirão no futuro para o registo de nós em Syncthing.

O exemplo seguinte foi feito entre dois dispositivos móveis (telefones) utilizando os seguintes modelos:

Dispositivo móvel #1: Modelo LG Q6

Dispositivo móvel #2: Modelo Samsung

comandos, abrir dois terminais separadamente:

Vamos começar por iniciar os serviços de rede Tor e os serviços de sincronização usando os seguintes

\$ tor

Após ter digitado o comando de arranque do programa de rede Tor, deve verificar se a execução foi bem sucedida, verificando se foi feita a 100%.

Executando o programa Tor num terminal Termux, verificamos que está a funcionar a 100%.

\$ tor



```

$ tor
May 23 23:00:30.932 [notice] Tor 0.4.3.5 running
on Linux with Libevent 2.1.11-stable, OpenSSL 1
.1.1g, Zlib 1.2.11, Liblzma 5.2.5, and Libzstd N
/A.
May 23 23:00:30.934 [notice] Tor can't help you
if you use it wrong! Learn how to be safe at htt
ps://www.torproject.org/download/download#warnin
g
May 23 23:00:30.939 [notice] Read configuration
file "/data/data/com.termux/files/usr/etc/tor/to
rrc".
May 23 23:00:30.970 [notice] I think we have 8 C
PUS, but only 6 of them are available. Telling T
or to only use 6. You can override this with the
NumCPUs option
May 23 23:00:30.973 [notice] Opening Socks liste
ner on 127.0.0.1:9050
May 23 23:00:30.974 [notice] Opened Socks liste
ner on 127.0.0.1:9050
May 23 23:00:30.000 [notice] Parsing GEOIP IPv4
file /data/data/com.termux/files/usr/share/tor/g
eoip.
May 23 23:00:32.000 [notice] Parsing GEOIP IPv6
file /data/data/com.termux/files/usr/share/tor/g
eoip6.
May 23 23:00:34.000 [notice] Bootstrapped 0% (st
arting); Starting
May 23 23:00:34.000 [notice] Starting with guard
context "default"
May 23 23:00:35.000 [notice] Bootstrapped 5% (co
nn); Connecting to a relay
May 23 23:00:36.000 [notice] Bootstrapped 10% (c
onn_done); Connected to a relay
May 23 23:00:36.000 [notice] Bootstrapped 14% (h
andshake); Handshaking with a relay
May 23 23:00:36.000 [notice] Bootstrapped 15% (h
andshake_done); Handshake with a relay done
May 23 23:00:36.000 [notice] Bootstrapped 20% (o
nehop_create); Establishing an encrypted directo
ry connection
May 23 23:00:36.000 [notice] Bootstrapped 25% (r
elay_selection)

```



```

$ tor
ps://www.torproject.org/download/download#warnin
g
May 24 01:33:32.982 [notice] Read configuration
file "/data/data/com.termux/files/usr/etc/tor/to
rrc".
May 24 01:33:33.007 [notice] I think we have 8 C
PUS, but only 6 of them are available. Telling T
or to only use 6. You can override this with the
NumCPUs option
May 24 01:33:33.010 [notice] Opening Socks liste
ner on 127.0.0.1:9050
May 24 01:33:33.010 [notice] Opened Socks liste
ner on 127.0.0.1:9050
May 24 01:33:33.000 [notice] Parsing GEOIP IPv4
file /data/data/com.termux/files/usr/share/tor/g
eoip.
May 24 01:33:34.000 [notice] Parsing GEOIP IPv6
file /data/data/com.termux/files/usr/share/tor/g
eoip6.
May 24 01:33:35.000 [notice] Bootstrapped 0% (st
arting)
May 24 01:33:37.000 [notice] Starting with guard
context "default"
May 24 01:33:38.000 [notice] Bootstrapped 5% (co
nn); Connecting to a relay
May 24 01:33:38.000 [notice] Bootstrapped 10% (c
onn_done); Connected to a relay
May 24 01:33:39.000 [notice] Bootstrapped 14% (h
andshake); Handshaking with a relay
May 24 01:33:39.000 [notice] Bootstrapped 15% (h
andshake_done); Handshake with a relay done
May 24 01:33:39.000 [notice] Bootstrapped 75% (e
nough_dirinfo); Loaded enough directory info to
build circuits
May 24 01:33:39.000 [notice] Bootstrapped 90% (a
p_handshake_done); Handshake finished with a rel
ay to build circuits
May 24 01:33:39.000 [notice] Bootstrapped 95% (c
ircuit_create); Establishing a Tor circuit
May 24 01:33:42.000 [notice] Bootstrapped 100% (d
one); Done

```

Depois executamos o comando syncthing:

Sincronização

Após executar este comando, abrirá uma página de administração no browser do nosso telefone se não abrir automaticamente, podemos ir a qualquer browser que tenhamos instalado onde normalmente navegamos na Internet e podemos colocar o seguinte l:

<http://127.0.0.1:8384>

Irá abrir o ecrã de administração da ferramenta que nos ajudará a sincronizar a nossa informação entre todos os nós (telefones) do sistema.



```
$ syncthing
[monitor] 04:02:07 INFO: Default folder created
and/or linked to new config
[start] 04:02:07 INFO: syncthing v1.5.0 "Fermium
Flea" (go1.14.2 android-arm) builder@6bdf862223
8a 2020-05-11 08:38:11 UTC
[start] 04:02:07 INFO: Generating ECDSA key and
certificate for syncthing...
[start] 04:02:08 INFO: Default folder created an
d/or linked to new config
[start] 04:02:08 INFO: Default config saved. Edi
t /data/data/com.termux/files/home/.config/synct
hing/config.xml to taste (with Syncthing stopped
) or use the GUI
[OWEPS] 04:02:08 INFO: My ID: OWEPSNA-6RZI6S7-SK
V0U65-V448C52-CXZOJI4-C3JC7HM-IPY4V35-JQAKPAW
[OWEPS] 04:02:09 INFO: Single thread SHA256 perf
ormance is 12 MB/s using crypto/sha256 (12 MB/s
using minio/sha256-simd).
[OWEPS] 04:02:11 INFO: Hashing performance is 11
.17 MB/s
[OWEPS] 04:02:11 INFO: Migrating database to sch
ema version 1...
```



Velocidad de descarga	0 B/s (0 B)
Velocidad de subida	0 B/s (0 B)
Estado Local (Total)	~0 B
Oyentes	3/3
Descubrimiento	4/5
Tiempo de funcionamiento	1m
Versión	v1.5.0, android (ARM)

Otros dispositivos

- Cambios recientes
- Añadir un dispositivo

Uma vez que temos o ecrã de administração "sincronização" aberto, procedemos ao registo do nó ou nós que queremos sincronizar a informação. É neste ponto que ocuparemos o programa que lê os códigos QR.

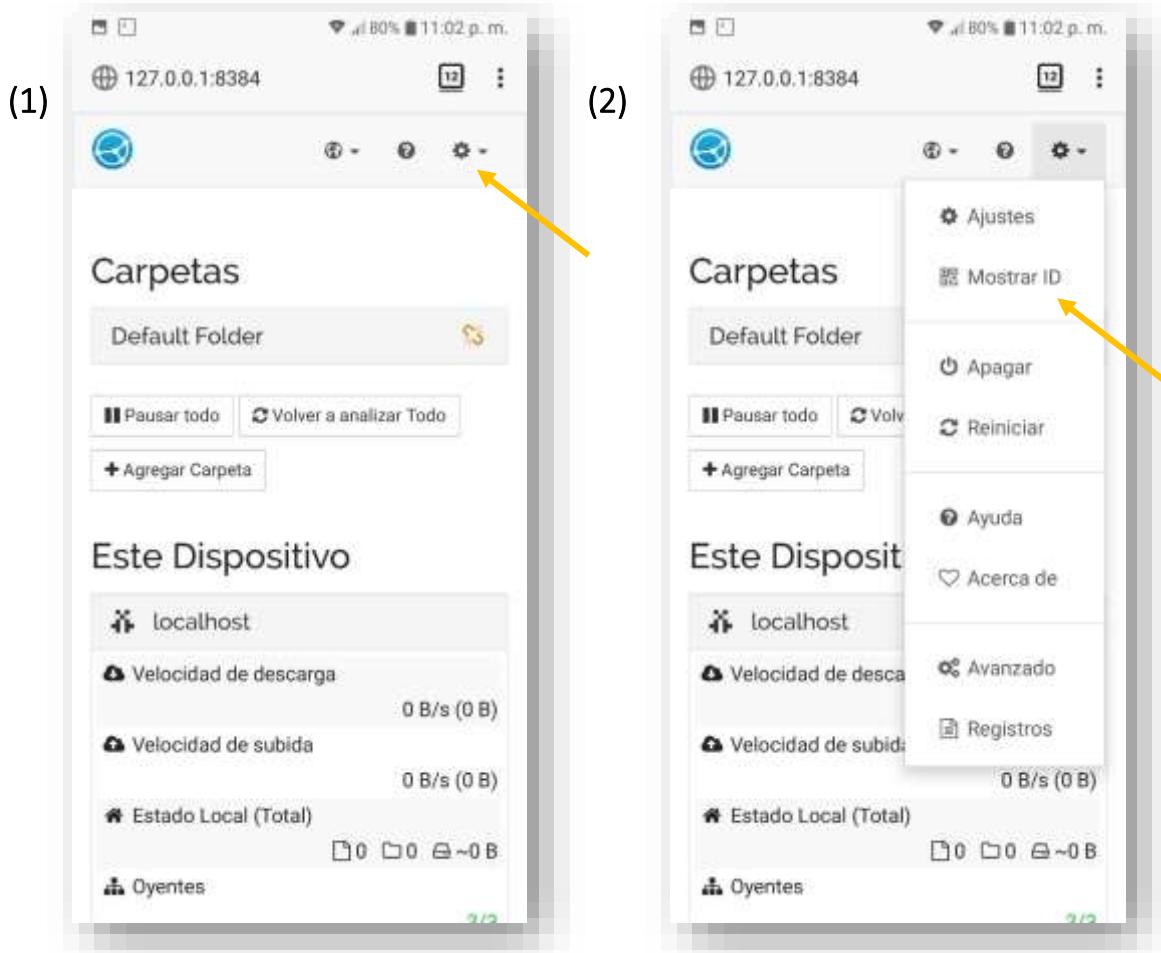
O programa de sincronização quando inicia pela primeira vez cria um identificador único do telefone que é composto por um grupo de oito conjuntos de caracteres alfanuméricicos em maiúsculas, este identificador (ID) é o que vamos registar no nó ou nós que queremos sincronizar a informação.

No nosso caso, a identificação telefónica LG Q6 terá de ser registada para o telefone Samsung e a identificação telefónica Samsung terá de ser registada para o LG Q6. Devem estar em ambos os telefones para funcionarem correctamente.

Iremos executar os passos do registo do telemóvel Samsung no telefone LG Q6.

Primeiro (1) no topo do ecrã de administração (browser de Internet) do telefone Samsung com sincronização, clicaremos no separador do menu no lado superior direito.

Segundo (2) passo veremos um menu, neste clicamos em "Mostrar ID" do Samsung.

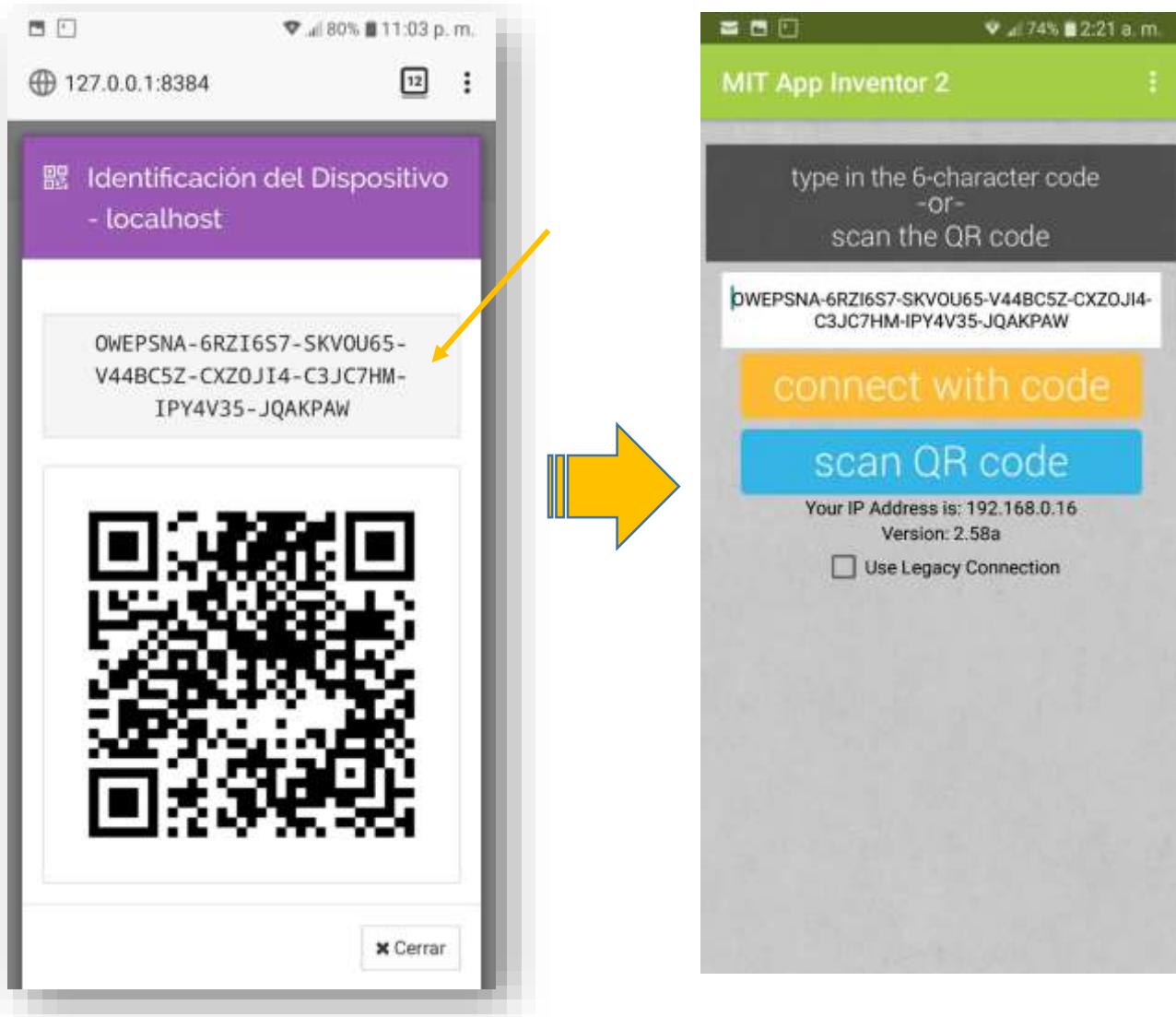


Ao clicar em "Mostrar ID" aparecerá o seguinte ecrã que é um código QR do telefone Samsung no nosso caso a ID que identifica o telefone é

OWEPSNA-6RZI6S7-SKVOU65-V44BC5Z-CXZOJI4-C3JC7HM-IPY4V35-JQAKPAW

Depois, no telefone LG Q6, o programa que nos ajudará a capturar este código QR do Samsung é o que sugerimos da aplicação App Inventor (opcional), embora no caso de não o termos também possamos simplesmente introduzi-lo manualmente quando iniciamos o registo no LG Q6, no entanto, para evitar qualquer erro sugerimos a sua utilização.

Usando o programa App Inventor instalado no telemóvel LG Q6 para capturar o código QR do telefone remoto Samsung que queremos sincronizar.



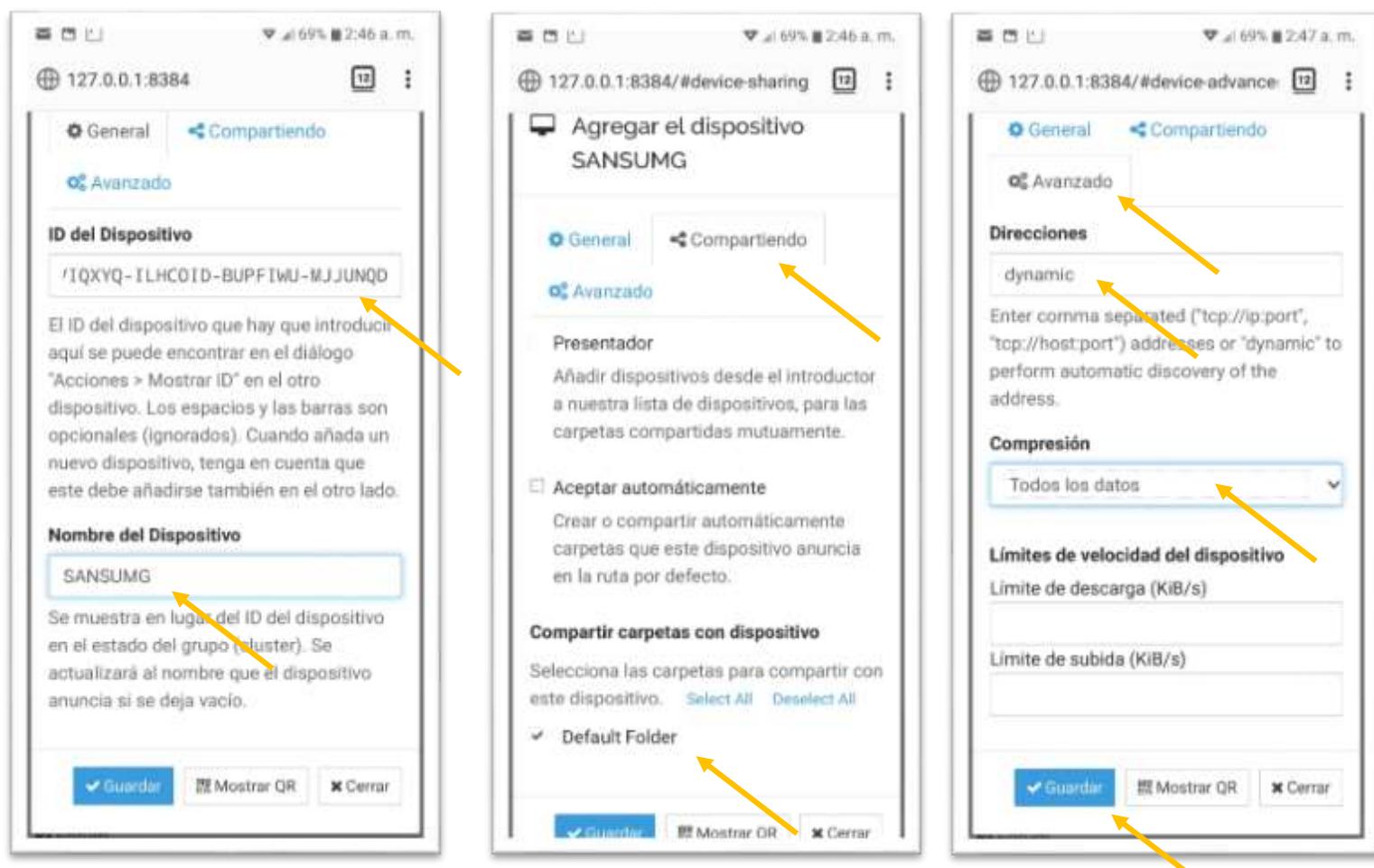
Depois copiamos para a prancheta o código QR no programa App inventor, clicando no código capturado, escolhemos "SELECT ALL" → "COPY".

Com esta informação, procedemos ao registo do Sansumg no LG Q6. No ecrã de administração passamos para a parte inferior onde diz "Outros dispositivos" e clicamos no botão "Adicionar um dispositivo", introduzimos o ID do Sansumg e damos-lhe um nome de registo.

Depois no separador superior "Partilhar" clicamos e neste marcamos a opção inferior na pasta "Default" com isto estaremos a partilhar um directório que foi criado por Default chamado Sync no nosso dispositivo.

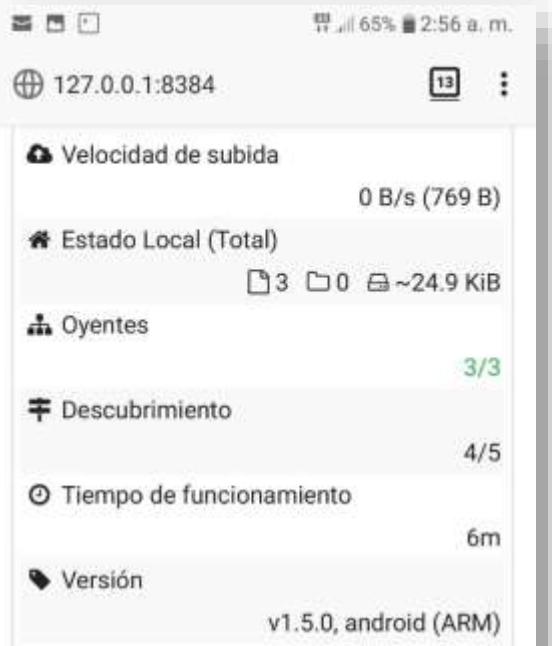
Depois, no topo, clicamos no separador "Avançado" e seleccionamos a opção de compressão de dados em "Todos os Dados".

Finalmente clicamos no botão inferior de guardar, terminamos o registo num nó, este mesmo processo deve ser feito no telefone remoto ou telefones que queremos sincronizar.



Ao guardar e registar em ambos os telefones, esperamos um período máximo de 30 segundos em que os dispositivos estão localizados e a ligação entre dispositivos aparecerá como boa dando uma confirmação em verde.

Dispositivo #1 LG Q6



Dispositivo #2 Sansumg



Otros dispositivos



Otros dispositivos



Toda a informação no directório Sync localizado no caminho /dados/dados/com.termux/file/home/Sync será sincronizada, quaisquer alterações serão copiadas e sincronizadas.

Configuração do sistema Peer to Peer com Syncthing automaticamente para utilização em Mini BlocklyChain.

Agora vamos fazer a configuração de uma forma automática, anteriormente fizemos o processo manual isto é útil se lidarmos com uma quantidade mínima de nós, no entanto, no caso de termos uma grande quantidade de nós seria ineficiente, por isso teremos a ferramenta SyncthingManager mais tarde para a configurar de uma forma automática utilizando comandos online automáticos.

Configuração da base de dados Redis para nós de modo (escravo).

Configuração do ficheiro `/data/data/com.termux/files/usr/etc/redis.conf` da base de dados Redis (**Slave**) para telemóveis Android.

Adicionar as seguintes alterações ou directivas ao ficheiro, guardar as alterações e iniciar o servidor Redis.

Primeiro, encontrar e remover o carácter # da linha **de escravo**. Esta directiva toma o endereço IP e a porta que utiliza para contactar com segurança o servidor mestre Redis, separados por um espaço. Por defeito, o servidor Redis ouve no 6379 na interface local, mas cada um dos métodos de segurança de rede altera o padrão de alguma forma para outros.

Os valores que utiliza dependerão do método que utilizou para proteger o tráfego da sua rede:

Rede isolada: utilizar o endereço IP da rede isolada e a porta Redis (6379) do servidor principal (por exemplo, escravo do endereço IP_ simples 6379).

stunnel ou spiped: utilizar a interface local (127.0.0.1) e a porta configurada para tunelar o tráfego

PeerVPN: Utilizar o endereço IP VPN do servidor mestre e a porta Redis normal.

O general iria mudá-lo:

escravo do ip_contact_server master_contact_port

Exemplo: **escravo de** 192.168.1.69 6379

masterauth a sua_senha_master_de_rede

Exemplo: **masterauth** sdfssdfsdfsd12WqE34Rfgthtdfd

necessite de passar a sua_senha_de_rede

Exemplo: **requirepass** asdsjdsh34sds67sdFGbbnh

Guardar e executar o servidor a partir do terminal Termux com o seguinte comando.

\$ redis-server redis.conf

Após a execução, podemos ver como se sincroniza com o servidor Windows 10 (**Master**).

Ficheiro de configuração redis.conf \$ redis-server redis.conf

The image consists of two side-by-side screenshots of the Termux terminal on an Android device. Both screenshots show a black background with white text and a standard Linux-style keyboard layout at the bottom.

Left Screenshot (Terminal 1):

```
$ pwd
/data/data/com.termux/files/usr/etc
$ ls
alternatives      inputrc    redis.conf
apt              krb5.conf
bash.bashrc       motd
bash_completion.d profile
dump.rdb          profile.d
tmux.conf
wgetrc
```

Right Screenshot (Terminal 2):

```
32672:S 31 May 2020 23:50:24.130 # Server initialized
32672:S 31 May 2020 23:50:24.131 * Loading RDB produced by version 6.0.1
32672:S 31 May 2020 23:50:24.131 * RDB age 27 seconds
32672:S 31 May 2020 23:50:24.131 * RDB memory usage when created 0.39 Mb
32672:S 31 May 2020 23:50:24.132 * DB loaded from disk: 0.001 seconds
32672:S 31 May 2020 23:50:24.132 * Ready to accept connections
32672:S 31 May 2020 23:50:24.132 * Connecting to MASTER 192.168.1.69:6379
32672:S 31 May 2020 23:50:24.136 * MASTER <-> REPLICATION sync started
32672:S 31 May 2020 23:50:24.159 * Non blocking connect for SYNC fired the event.
32672:S 31 May 2020 23:50:24.166 * Master replied to PING, replication can continue...
32672:S 31 May 2020 23:50:24.236 * Partial resynchronization not possible (no cached master)
32672:S 31 May 2020 23:50:24.266 * Full resync from master: 8ea52fe3c02ae241292f0dcbb823b8febcb09b784:0
32672:S 31 May 2020 23:50:24.349 * MASTER <-> REPLICATION sync: receiving 578 bytes from master to disk
32672:S 31 May 2020 23:50:24.353 * MASTER <-> REPLICATION sync: Flushing old data
32672:S 31 May 2020 23:50:24.353 * MASTER <-> REPLICATION sync: Loading DB in memory
32672:S 31 May 2020 23:50:24.354 * Loading RDB produced by version 5.0.7
32672:S 31 May 2020 23:50:24.354 * RDB age 0 seconds
32672:S 31 May 2020 23:50:24.354 * RDB memory usage when created 1.84 Mb
32672:S 31 May 2020 23:50:24.355 * MASTER <-> REPLICATION sync: Finished with success
```

Instalação de ferramenta de gestão de sincronização para nós. Procedemos à instalação do **SyncthingManager**.

Primeiro instalamos o que necessitará para que o SyncthingManager funcione correctamente.

\$ apt instalar Python

Instalação de \$pip3 -upgrade pip

\$ npm instalar syncthingmanager

A ferramenta SyncthingManager irá ajudar-nos a sincronizar "peer to peer" de forma automática e não manualmente para nós novos e existentes.

```
$ apt install python
Reading package lists... Done
Building dependency tree
Reading state information... Done
python is already the newest version (3.8.3),
0 upgraded, 0 newly installed, 0 to remove and 0
not upgraded.
$ 

$ pip3 install --upgrade pip
Requirement already up-to-date: pip in /data/dat
a/com.termux/files/usr/lib/python3.8/site-pac
kes (20.1.1)
$ 

$ pip3 install syncthingmanager
Requirement already satisfied: syncthingmanager
in /data/dat
a/com.termux/files/usr/lib/python3.8/site-p
ackages (0.1.0)
Requirement already satisfied: syncthing in /dat
a/com.termux/files/usr/lib/python3.8/site-p
ackages (from syncthingmanager) (2.3.1)
Requirement already satisfied: python-dateutil==2.6.1
in /data/dat
a/com.termux/files/usr/lib/python3.8/site-p
ackages (from syncthing->syncthingm
anager) (2.6.1)
Requirement already satisfied: requests==2.18.4
in /data/dat
a/com.termux/files/usr/lib/python3.8/site-p
ackages (from syncthing->syncthingm
anager) (2.18.4)
Requirement already satisfied: six>=1.5 in /data
/d
a/com.termux/files/usr/lib/python3.8/site-p
ackages (from python-dateutil==2.6.1->syncthing->
syncthingmanager) (1.15.0)
Requirement already satisfied: chardet<3.1.0,>=3
.0.2 in /data/dat
a/com.termux/files/usr/lib/python3.8/site-p
ackages (from requests==2.18.4->sync
thing->syncthingmanager) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in
/d
a/d
a/com.termux/files/usr/lib/python3.8/site-p
ackages (from requests==2.18.4->syncthing->
syncthingmanager) (2.6)
Requirement already satisfied: certifi>=2017.4.1
7 in /data/dat
a/com.termux/files/usr/lib/python3
.8/site-p
ackages (from requests==2.18.4->syncthing->syncthingm
anager) (2020.4.5.1)
Requirement already satisfied: urllib3<1.23,>=1
.21.1 in /data/dat
a/com.termux/files/usr/lib/python3.8/site-p
ackages (from requests==2.18.4->sync
thing->syncthingmanager) (1.22)
$ 
```

17. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).

App Inventor é um ambiente de desenvolvimento de software criado pelo Google Labs para construir aplicações para o sistema operativo Android. O utilizador pode, visualmente e a partir de um conjunto de ferramentas básicas, ligar uma série de blocos para criar a aplicação. O sistema é gratuito e pode ser facilmente descarregado a partir da web. As aplicações criadas com App Inventor são muito fáceis de criar porque não é necessário qualquer conhecimento de qualquer linguagem de programação.

Todos os ambientes actuais que utilizam a tecnologia Blockly, tais como AppyBuilder e Thunkable, entre outros, têm a sua versão gratuita, a sua forma de utilização pode ser através da Internet nos seus diferentes sites ou pode também ser instalada em casa.

Os blocos que compõem a arquitectura Mini BloclyChain foram testados em App inventor e AppyBuilder, mas devido à sua optimização de código, devem trabalhar nas outras plataformas.

Versões online:

App Inventor.

<https://appinventor.mit.edu/>

AppyBuilder.

<http://appybuilder.com/>

Aconchegável.

<https://thunkable.com/>

Versão a ser instalada no seu computador (PC):

<https://sites.google.com/site/aprendeappinventor/instala-app-inventor>

Ambiente para programadores de blocos Blockly.

<https://editor.appybuilder.com/login.php>

18. O que é a Prova de Quantum (PQu)?

PoQu. - "Proof of Quantum" é um algoritmo de consenso desenvolvido para Mini BlocklyChain, este teste é uma variante do Teste de Trabalho (PoW) que funciona da seguinte forma.

O Teste de Quantum (PoQu) no arranque é executado com o mesmo algoritmo que o "Teste de Trabalho" (PoW) baseia-se em colocar o processador do dispositivo (PC, Servidor, Tablet ou Telemóvel) a trabalhar para obter uma sequência de caracteres que é um puzzle matemático chamado "hash".

Lembre-se que um "hash" é um algoritmo ou processo matemático que ao introduzir uma frase ou algum tipo de informação digital tal como ficheiros de texto, programa, imagem, vídeo, som ou outro tipo de informação diversa nos dá como resultado um carácter alfanumérico que representa a assinatura digital que a representa de uma forma única e irrepetível dos dados, o algoritmo hash é unidireccional, isto significa que quando se introduz um dado para obter a sua assinatura "hash" o seu processo inverso não pode ser executado, tendo uma assinatura "hash" não podemos saber que informação foi obtida esta propriedade dá-nos uma vantagem de segurança para processar a informação que enviamos através da Internet. Como funciona? Imagine enviar qualquer tipo de informação através de canais não seguros e acompanhá-la com o seu respectivo "hash de origem", o receptor ao receber a informação pode obter o "hash" da informação recebida chamar-lhe-emos "hash de destino" e verificá-lo com o "hash de origem" se ambos os "hashes" forem os mesmos podemos confirmar que a informação não foi alterada no canal que foi enviado, é apenas um exemplo onde este tipo de processo de segurança da informação é actualmente utilizado.

Actualmente existem diferentes tipos de algoritmos ou processos de haxixe que diferem no nível de segurança. Os mais utilizados ou conhecidos são: MD5, SHA256 e SHA512.

Exemplo de SHA256:

Temos uma cadeia ou frase como se segue: "A Mini BlocklyChain é modular.

Se aplicarmos um hash SHA256 ao hash anterior, este dar-nos-á o hash seguinte.

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db8

A cadeia alfanumérica acima é a assinatura que representa a frase no exemplo acima

Para mais exemplos, podemos utilizar o sitio na Internet:

<https://emn178.github.io/online-tools/sha256.html>

No caso do algoritmo "Test Work" (PoW), funciona usando a potência de computação para obter um hash pré-definido.

Imaginemos que temos o "hash" anterior que retirámos da cadeia "Mini BlocklyChain é modular".

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db8

A este "hash" no seu início colocamos o parâmetro de dificuldade que é simplesmente colocar zeros "0" no início, ou seja, se dissermos que a dificuldade é de 4 terá "**0000**" + "hash" a isto chamar-lhe-emos "hash de semente".

0000 f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db8

Agora tendo em conta que sabemos a informação de entrada que é a cadeia: "Mini BlocklyChain é modular" adicionamos no fim da cadeia um número que começa em zero "0" e retiramos o seu hash a este, chamar-lhe-emos "hash nonce":

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db80

Temos hash nonce:

7529f3ad273fc8a9eff12183f8d6f886821900750bb6b59c1504924dfd85a7c8

Depois realizamos uma comparação do novo "hash nonce" com o "hash seed" se forem iguais o nó que primeiro encontrar a igualdade ganhará a execução do processamento da transacção actual. Como podemos ver, este processo baseia-se na probabilidade e força computacional do dispositivo que dá ao teste "Prova de Trabalho" uma equidade consensual para todos os nós.

Se o "hash semente" não coincide com o "hash nonce", a dificuldade é aumentada em um e o "hash nonce" é novamente removido, o número que está a ser aumentado é chamado de "nonce", é comparado com o "hash semente" até que coincidam ou sejam o mesmo.

Como podemos ver o número "nonce" ou aumento é o que ajudará a obter o "hash" da igualdade.

Baseado no algoritmo "Test of Work" (PoW), o algoritmo Test of Quantum (PoQu) baseia-se na obtenção do número "nonce" como o PoW e utilizando um nível mínimo de dificuldade que vai de 1 a 5, isto serve apenas para o dispositivo móvel ganhar o direito de ser um candidato para ganhar o consenso.

O Teste Quantum (PoQu), é activado quando o telemóvel termina o PoW mínimo e ganha o passe para obter um número de probabilidade no sistema QRNG.

O QRNG (Quantum Random Number Generator) é um Gerador de Números Quânticos Aleatórios, este sistema baseia-se na geração de números verdadeiramente aleatórios com base na mecânica quântica é o sistema mais seguro hoje em dia para gerar tais números. Para mais detalhes ver Anexo "Computação Quântica com OpenQbit".

A Mini BlocklyChain pode implementar ambos os tipos de concessões mínimas PoW e PoQu.

O teste PoQu baseia-se na obtenção do número "nonce" este número no teste PoQu é conhecido como "Número Mágico" com isto o sistema "Peer to Peer" confirmará se o número está correcto e depois será obtido um número aleatório com o pool de servidores QRNG. Este número aleatório será registado em todos os nós, será criada uma lista contendo **((Soma do nó /2)) +1** e desta lista será escolhida a que tiver a maior percentagem de probabilidade de ser o candidato vencedor do consenso (PoQu) e esta executará a fila de transacções em curso.

O algoritmo PoQu também utiliza testes **NIST** (National Institute of Standards and Technology) para nos assegurar que os números aleatórios no QRNG são números verdadeiramente aleatórios.

<https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf>

Na Mini BlocklyChain implementámos um bloco para PoW e um bloco para PoQu.

Estes blocos utilizam um tipo de hash: SHA256 para uso gratuito, para uso comercial tem um SHA512 e outros tipos de hash conforme necessário.

Para mais detalhes sobre o conceito de HASH, ver:

https://es.wikipedia.org/wiki/Funcion_hash

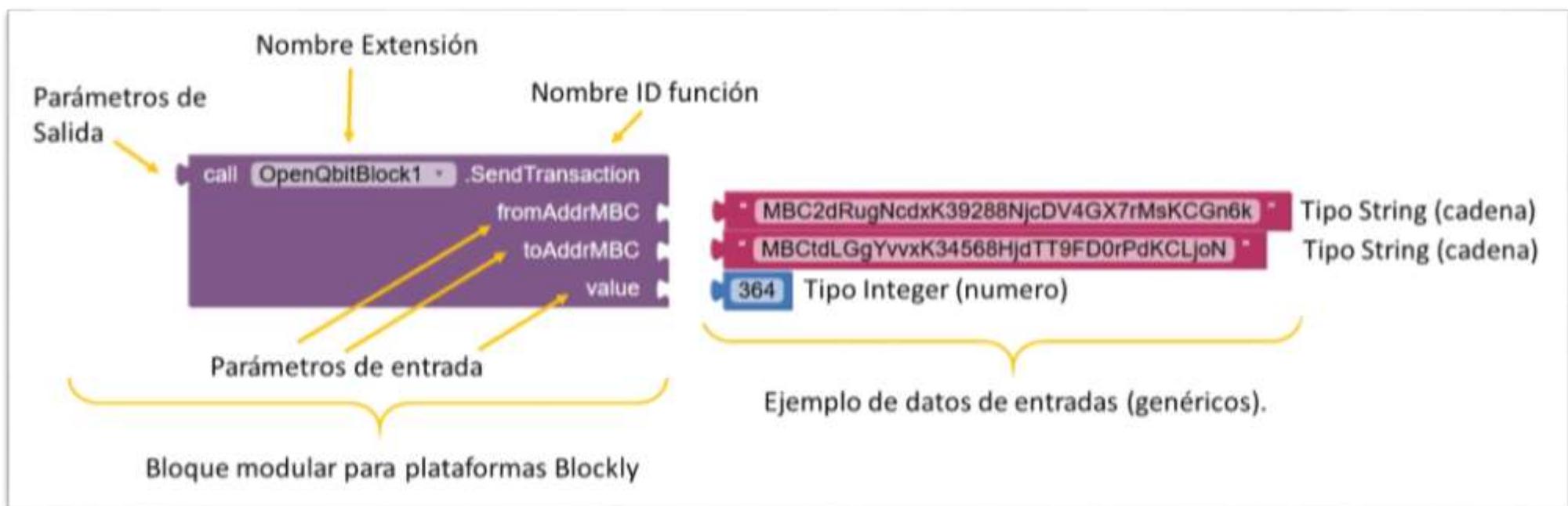
NOTA: O Teste de Trabalho (PoW) utilizado em telemóveis só pode utilizar uma dificuldade máxima de 5, uma vez que o processamento matemático destes dispositivos não é dedicado como servidores ou PCs. Apenas utilizamos o algoritmo PoW para obter a oportunidade de obter o seu passe ou permissão para entrar no sistema Quantum Random Number Generator (QRNG) e com ele para executar o algoritmo Quantum Random Number Generator (PoQu).

Nos telemóveis não utilizar uma dificuldade máxima de 5, pois o sistema pode bloquear e não responder adequadamente.

19. Definição e utilização de blocos em Mini BlocklyChain

Começaremos por explicar a distribuição dos dados que todos os blocos terão, a sua sintaxe de utilização e configuração.

No exemplo seguinte podemos ver um bloco modular e os seus parâmetros de entrada e saída, assim como os tipos de dados de entrada, estes dados podem ser do tipo String (cadeia de caracteres) ou Inteiro (inteiro ou decimal). Mostramos como é utilizado e configuramos para o seu bom funcionamento.



Cada bloco de módulos terá a sua descrição e será nomeado caso tenha alguma dependência(s) obrigatória(s) ou opcional(s) de outros blocos utilizados como parâmetros de entrada, o processo de integração será anunciado.

Bloco para criar uma lista de cordas temporária numa matriz pré-definida chamada internamente "cadeia" - (AddHash).



Parâmetros de entrada: **bloco** <string>

Parâmetros de saída: Não aplicável.

Descrição: Bloco para armazenar um conjunto temporário de hashes ou cordas num conjunto pré-definido chamado internamente "cadeia".

Bloco para criar matrizes de valores chave (**Integer-Integer**) - (AddKeyValueArrayItol)

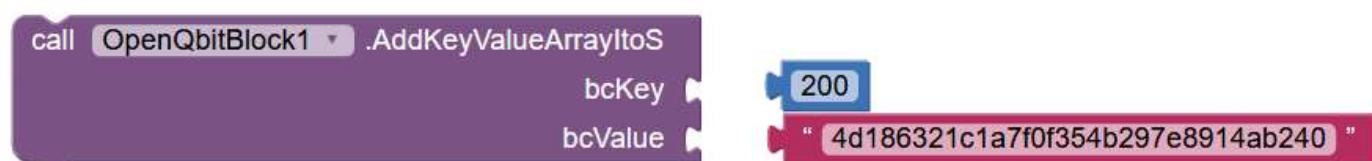


Parâmetros de entrada: **bcKey** <Integer>, **bcValor** > Integer

Parâmetros de saída: Retorna os valores introduzidos na entrada.

Descrição: É um acordo temporário de valor chave, é predefinido com o nome interno "Itol_UTXO", isto é útil para processar transacções UTXO.

Bloco para criar matrizes de valor chave (**Integer-String**) - (AddKeyValueArrayItoS)



Parâmetros de entrada: **bcKey** <Integer>, **bcValue** < String>

Parâmetros de saída: Retorna os valores introduzidos na entrada.

Descrição: É um acordo temporário de valor chave, é predefinido com o nome interno "ItoS_UTXO", isto é útil para processar transacções UTXO.

Bloco para criar matrizes de valores chave (**String-Integer**) - (**AddKeyValueArrayItol**)



Parâmetros de entrada: **bcKey** <Calça>, **bcValor**>Integer>

Parâmetros de saída: Retorna os valores introduzidos na entrada.

Descrição: É um acordo temporário de valor chave, é predefinido com o nome interno "Stol_UTXO", isto é útil para processar transacções UTXO.

Bloco para criar matrizes de valor chave (**String-String**) - (**AddKeyValueArrayStoS**)

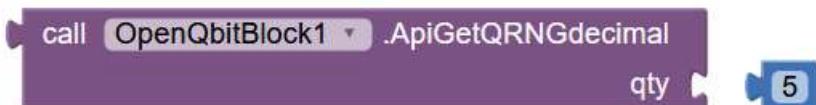


Parâmetros de entrada: **bcKey** <String>, **bcValue**>String>

Parâmetros de saída: Retorna os valores introduzidos na entrada.

Descrição: É um acordo temporário de valor chave, é predefinido com o nome interno "StoS_UTXO", isto é útil para processar transacções UTXO.

Bloco para gerar números quânticos decimais aleatórios - (**ApiGetQRNGdecimal**)



Parâmetros de entrada: **qty** < Integer >

Parâmetros de saída: Dá a quantidade "qty" de números quânticos decimais aleatórios introduzidos nos números de entrada estão dentro do intervalo de 0 e 1 no formato JSON.

Exemplo:

qty = 5; saída: {"resultado": [0,5843012986202495, 0,7746497687824652, 0,05951126805960929, 0,1986079055812694, 0,03689783439899279]}

Descrição: Gerador de números aleatórios quânticos (QRNG) API

Bloco para gerar números quânticos decimais aleatórios - (**ApiGetQRNGdecimal**)



Parâmetros de entrada: **qty <Integer>**, min <Integer>, max <max>

Parâmetros de saída: Dá a quantidade "qty" de inteiros quânticos aleatórios introduzidos na entrada os números estão dentro do intervalo de min e max no formato JSON.

Exemplo:

qty = 8, min = 1, max = 100; saída: {"resultado": [3, 53, 11, 2, 66, 44, 9, 78]}

Descrição: Gerador de números aleatórios quânticos (QRNG) API

Bloco de hastes "SHA256" para cordas de caracteres (**ApplySha256**).



Parâmetros de entrada: **entrada <Calça>**

Parâmetros de saída: Calcula o hash "SHA256" de uma cadeia de caracteres.

Exemplo:

Input = "Mini BlocklyChain é modular"

saída: f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db8

Descrição: Função para remover haxixe "SHA256". Sha ou SHA referem-se a: Algoritmo de Hash Seguro, um conjunto de funções de hash concebidas pela Agência de Segurança Nacional dos Estados Unidos. O SHA256 utiliza um algoritmo de 256 bits.

Bloco para limpar a "cadeia" predefinida da matriz interna (**ClearBlockList**).

call OpenQbitBlock1 .ClearBlockList

Parâmetros de entrada e saída: Não aplicável

Descrição: Eliminar todos os elementos que têm a "cadeia" de disposição temporal interna.

Bloco para limpar a matriz interna predefinida (**Integer-Integer**) - (**ClearItoltol**).

call OpenQbitBlock1 .ClearItoltol

Parâmetros de entrada e saída: Não aplicável

Descrição: Eliminar todos os elementos que têm a disposição temporária interna "Itoltol_UTXO".

Bloco para limpar a matriz interna predefinida (**Integer-String**) - (**ClearItoS**).

call OpenQbitBlock1 .ClearItoS

Parâmetros de entrada e saída: Não aplicável

Descrição: Eliminar todos os elementos que têm a disposição temporária interna "ItoS_UTXO".

Bloco para limpar a matriz interna predefinida (**String-Integer**) - (**ClearStol**).

call OpenQbitBlock1 .ClearStol

Parâmetros de entrada e saída: Não aplicável

Descrição: Eliminar todos os elementos que têm a disposição temporária interna "Stol_UTXO".

Bloco para limpar a matriz interna predefinida (**String-String**) - (**ClearStoS**).

call OpenQbitBlock1 .ClearStoS

Parâmetros de entrada e saída: Não aplicável

Descrição: Eliminar todos os elementos que têm a disposição temporária interna "StoS_UTXO".

Bloco para descodificar uma corda para Base10. (**DecodeBase58**)



Parâmetros de entrada: **entrada<Calça>**

Parâmetros de saída: Fornece o fio original que foi utilizado no bloco (**EncodeBase58**).

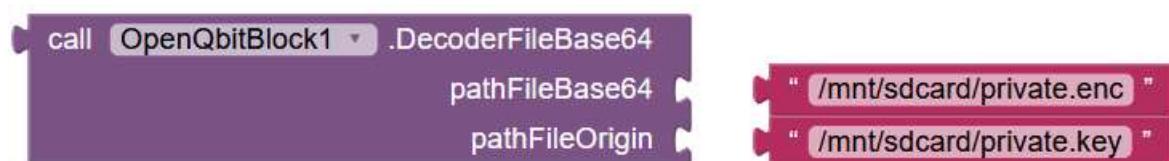
Exemplo:

Entrada = oBRN8Aj67eJsbRGHfNSF9PTdZYGandVWrwMW7mTX

Saída: "Mini BlocklyChain é modular".

Descrição: Converte uma cadeia Base58 para o texto original dado no bloco (**EncodeBase58**)

Bloco para descodificar um ficheiro com algoritmo Base64 (**DecoderFileBase64**).



Parâmetros de entrada: **pathFileBase64 <String>**, **pathFileOrigin <String>**

Parâmetros de saída: ficheiro fonte que foi introduzido no bloco (**EncoderFileBase64**)

Descrição: Um ficheiro Base64 é convertido para o ficheiro original que foi inserido no bloco (**EncoderFileBase64**).

Bloco para saber se a "cadeia" interna pré-definida está vazia. (**EmptyBlockList**)



Parâmetros de entrada: Não aplicável.

Parâmetros de saída: Retorna "Verdadeiro" se estiver vazio ou retorna "Falso" se tiver dados.

Descrição: Bloco para perguntar se a "cadeia" interna temporária pré-definida tem elementos.

Bloco para codificar uma cadeia de caracteres para Base58. (**EncodeBase58**).



Parâmetros de entrada: **entrada<Calça>**

Parâmetros de saída: Fornece o fio original que foi utilizado no bloco (**EncodeBase58**).

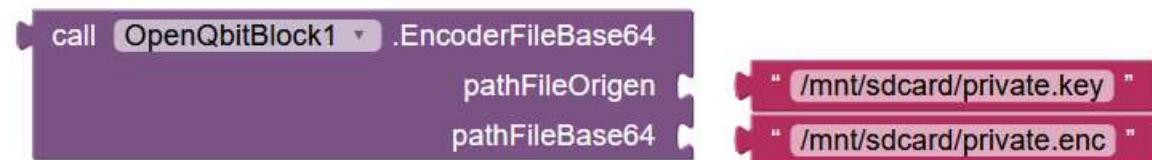
Exemplo:

Input = "Mini BlocklyChain é modular".

A nossa produção: oBRN8Aj67yJsbRGHfNSF9PTdZYGyVWrwMW7mTX

Descrição: Converte uma corrente mamária numa corrente em Bae58. O algoritmo Base58 é um grupo de esquemas de codificação binária a texto utilizado para representar grandes inteiros como texto alfanumérico, introduzido por Satoshi Nakamoto para uso com Bitcoin.

Bloco para codificar um ficheiro com algoritmo Base64 (**EncoderFileBase64**).



Parâmetros de entrada: **pathFileOrigin <String>**, **pathFileBase64 <String>**

Parâmetros de saída: ficheiro codificado com base64.

Descrição: Converte um ficheiro fonte de qualquer formato para um ficheiro Base64. Os nomes dos ficheiros podem ser arbitrários e escolhidos pelo utilizador.

Gerador de blocos de endereços de utilizadores (**GenerateAddrBitcoin**).



Unidade obrigatória: Bloco (**ApiGetQRNGinteger**),

Dependências (opcional): extensão **OpenQbitFileEncription**, extensão **OpenQbitFileDecryption** e extensão **OpenQbitSQLite**.

Parâmetros de entrada: **qrng** < dependência obrigatória>

Parâmetros de saída: endereço de transacção alfanumérico de 34 caracteres e endereço de utilização da KeyStore

Descrição: Bloco para criar um novo endereço de transacção genérico Bitcoin para gerador de chave de utilizador e privado (Assinatura digital para envio de transacções) e chave pública (Endereço público para realização de transacções). Este gerador chave é basicamente o gerador de endereços para utilização numa carteira digital ou comumente chamado "Carteira".

Este bloco é utilizado em conjunto com os blocos Quantum Random Number Generator (QRNG) e os dois parâmetros de saída devem ser inseridos na KeyStore.

KeyStore é uma base de dados que armazena chaves privadas em formato hexadecimal:

Exemplo de um endereço hexadecimal que está armazenado na KeyStore

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

Esta base é utilizada apenas pelo utilizador local e a informação é encriptada, este processo é através da utilização das extensões **OpenQbitSQLite** e **OpenQbitAESEncryption** e **OpenQbitAESDecryption** de encriptação de informação.

Para criar uma KeyStore ver o apêndice "Criar uma KeyStore". Os endereços gerados utilizam o mesmo algoritmo de endereço Bitcoin, com o identificador inicial de endereço Bitcoin a ser "1".

Os endereços gerados pelo bloco (**GenerateAddrBitcoin**) são 34 caracteres alfanuméricos são compostos por 33 caracteres alfanuméricos e 1 do identificador Bitcoin, como se segue:

12dRugNcdxK39288NjcDV4GX7rMsKCGn6k

Bloco Gerador de Endereço de Utilizador (**GenerateAddrEthereum**).



Dependência obrigatória: Obter uma ficha em: <https://accounts.blockcypher.com/signup>

Dependências (opcional): extensão **OpenQbitFileEncription**, extensão **OpenQbitFileDecryption** e extensão **OpenQbitSQLite**.

Parâmetros de entrada: **token** < dependência obrigatória - String>

Parâmetros de saída: 40 endereços de transacção de caracteres alfanuméricos não incluem o indicador inicial Ethereum "0x", que nos daria os 42 caracteres de um endereço comum. Também devolve a chave pública e a chave privada.

Exemplo:

```
{
  "privado": "227ac59f480131272003c2d723a7795ebd3580acaab62b5c537989e2ce4e08ef",
  "público":
  "04e2d55ebccd32a7384e096df559cc36b856c64a16e5b402e10585dc3ea055672aafa84df8
  a859531570a650a8ab1e7a22949100efa1aaa5f072c035551cac1ce",
  "endereço": "14e150399b0399f787b4d6fe30d8b251375f0d66"}
```

Descrição: Bloco para criar um novo endereço de transacção para o utilizador e gerador de chaves privadas (assinatura digital para enviar transacções) e chave pública (endereço público para fazer transacções). Este gerador de chaves é uma API externa e deve ter uma ligação Wifi ou móvel, é basicamente o gerador de endereços para o utilizar numa carteira digital ou comumente chamada "Carteira".

Pode criar umaKeyStore é uma base de dados que armazena chaves privadas em formato hexadecimal, Ver Apêndice "Criação de KeyStore".

Exemplo de um endereço hexadecimal que está armazenado na KeyStore

0x14e150399b0399f787b4d6fe30d8b251375f0d66

A chave privada é utilizada apenas pelo utilizador local e a informação é encriptada, este processo é através da utilização da extensão OpenQbitSQLite e OpenQbitAESEncryption e OpenQbitAESDecryption de extensões de encriptação de informação.

Bloco Gerador de Endereço de Utilizador (**GenerateAddrMiniBlocklyChain**).



Unidade

obrigatória: Bloco (**ApiGetQRNGinteger**),

Dependências (opcional): extensão **OpenQbitFileEncription**, extensão **OpenQbitFileDecryption** e extensão **OpenQbitSQLite**.

Parâmetros de entrada: **qrng < dependência obrigatória>**

Parâmetros de saída: 36 endereço de transacção de caracteres alfanuméricos e endereço de utilização da KeyStore. Método do bloco de revisão (**PairKeysMBC**).

Descrição: Bloco para criar um novo endereço de transacção para o utilizador e gerador de chaves privadas (assinatura digital para enviar transacções) e chave pública (endereço público para fazer transacções). Este gerador de chaves é basicamente o gerador de endereços para o utilizar numa carteira digital ou normalmente chamado "Carteira".

Este bloco é utilizado em conjunto com os blocos Quantum Random Number Generator (QRNG) e os dois parâmetros de saída devem ser inseridos na KeyStore.

KeyStore é uma base de dados que armazena chaves privadas em formato hexadecimal:

Exemplo de um endereço hexadecimal que está armazenado na KeyStore

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

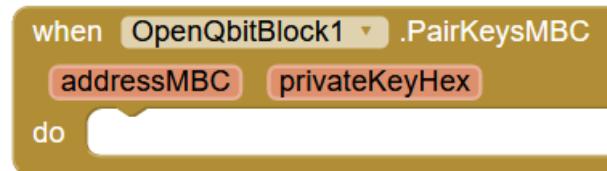
Esta base é utilizada apenas pelo utilizador local e a informação é encriptada, este processo é através da utilização das extensões **OpenQbitSQLite** e **OpenQbitAESEncryption** e **OpenQbitAESDecryption** de encriptação de informação.

Para criar uma KeyStore ver o apêndice "Criar uma KeyStore". Os endereços gerados utilizam o mesmo algoritmo de endereço bitcoin, a única diferença é que o identificador de endereço bitcoin que é "1" ou "3" é alterado para o identificador Mini BlocklyChain "MBC".

Os endereços gerados pelo bloco (**GenerateAddrMiniBlocklyChain**) são 36 caracteres alfanuméricos e são compostos por 33 caracteres alfanuméricos e 3 letras maiúsculas do Mini BlocklyChain identificador "MBC" como se segue:

Mbc2dRugNcdxK39288NjcDV4GX7rMsKCGn6k

O método de bloco (**PairKeysMBC**) é a saída em bloco (**GenerateAddrMiniBlocklyChain**).



Parâmetros de entrada: Não aplicável.

Parâmetros de saída: **addressMBC <String>**, **privateKeyHex <String>**.

Descrição: Fornecido endereço público de utilizador em formato Mini BlocklyChain e chave privada em formato hexadecimal.

exemplo:

endereçoMBC: MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k

privateKeyHex:

024C8E05018319ARD4BBB04E184C307BFF115976A05F974C7D945B5151E490ERD

Gerador de blocos de endereços de utilizadores (**GenerateKeyValuePair**).



Unidade obrigatória: Bloco (**ApiGetQRNGinteger**),

Dependências (opcional): extensão **OpenQbitFileEncription**, extensão **OpenQbitFileDecryption** e extensão **OpenQbitSQLite**.

Parâmetros de entrada: **qrng < dependência obrigatória>**

Parâmetros de saída: Fornece a chave pública e a chave primária para o utilizador local

Descrição: Gera chaves públicas e primárias usando algoritmo ECC (1) e com formato hexadecimal.

- (1) A criptografia de curvas elípticas (ECC) é uma variante da criptografia assimétrica ou de chave pública baseada na matemática das curvas elípticas.

Bloco para assinar transacções (**GenerateSignature**).

```
call OpenQbitBlock1 .GenerateSignature
```

Dependência necessária: Bloco (**GenerateKeyPairs**), Bloco (**SenderLoadKeyPair**), Bloco (**RecipientLoadKeyPair**). Prefácio destes blocos antes da sua utilização.

Parâmetros de entrada: < Dependências de verificação obrigatórias>

Parâmetros de saída: Sem saída.

Descrição: Gera uma assinatura digital do Remetente aplicada ao bem enviado na transacção para o Destinatário, esta assinatura será confirmada quando a transacção estiver a ser processada por algum nó da rede, com esta assinatura o sistema Mini BlocklyChain assegura que o bem não foi modificado no bem enviado e que está em conformidade com a relação remetente-receptor e não é desviado ou aplicado para outro endereço digital.

Bloco para obter o saldo total de activos de um utilizador (**GetBalance**).

```
call OpenQbitBlock1 .GetBalance
```

```
fromAddrMBC
```

```
"MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k"
```

Dependência obrigatória: Bloco (**ConnectorTransactionTail**).

Parâmetros de entrada: **daTransTail <Array String>**, <Dependências obrigatórias das Placas>

Parâmetros de saída: Verifica as despesas de entrada e saída de activos para cada transacção.

Descrição: Verifica o saldo a aprovar se o utilizador tem bens a enviar ou se os bens que recebe são adicionados ao seu saldo.

Bloco para obter q22 do telemóvel. (**GetDeviceID**)

```
call OpenQbitBlock1 .GetDeviceID
```

Parâmetros de entrada: **Não aplicável**

Parâmetros de saída: Fornece o identificador interno do telemóvel.

Descrição: Fornece o identificador de telemóvel IMEI único para cada dispositivo.

Bloco para remover o hash RIPEMD-160 de uma corda. (**Ripemd160**)



Parâmetros de entrada: **str <String>**

Parâmetros de saída: Calcula o hash "RIPEMD-160" de uma cadeia de caracteres.

Exemplo:

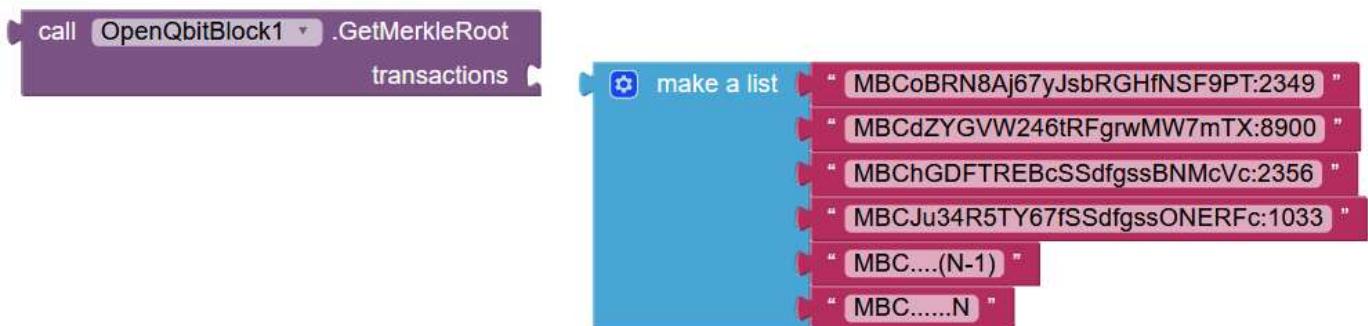
Input = "Mini BlocklyChain é modular"

saída: ae29436e4b8ea8ed6143f3f92380dfa2f4f47336

Descrição: Obter o hash "RIPEMD-160". Este hash é utilizado no processo de geração de um endereço válido para Bitcoin e Mini BlocklyChain.

RIPEMD-160 (acrônimo de RACE Integrity Primitives Evaluation Message Digest) é um algoritmo de digestão de mensagens de 160 bits (e função de hash criptográfico).

Bloco para calcular a árvore de Merkler. (**GetMerkleRoot**)



Parâmetros de entrada: **transacções <Array String>**

Parâmetros de saída: Fornece um tipo de hash "SHA256".

Exemplo:

Input = fila de transações, um arranjo de todas as transações a serem processadas.

saída: b4a44c42b6070825f763cd118d6ab49a8e80bbb7cdc0225064f8e042b94196bd

Descrição: Obter o hash "SHA256" usando o algoritmo de árvore da Merkle

Uma Merkle Hash Tree é uma estrutura de árvore de dados binária ou não binária na qual cada nó que não é uma folha é marcado com o hash da concatenação das etiquetas ou

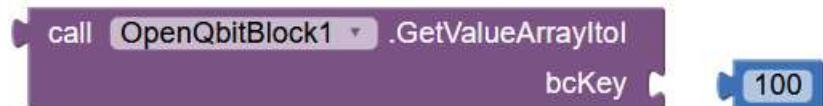
valores (para nós de folha) dos seus nós filhos. Estas são uma generalização de listas de hash e cadeias de hash.

Permite ligar um grande número de dados separados a um único valor de hash, o hash do nó de raiz da árvore. Isto proporciona um método seguro e eficiente de verificação do conteúdo de grandes estruturas de dados. Nas suas aplicações práticas, o hash do nó raiz é normalmente assinado para assegurar a sua integridade e que a verificação é totalmente fiável. Demonstrar que um nó de folha faz parte de uma determinada árvore de hash requer uma quantidade de dados proporcional ao logaritmo do número de nós na árvore.

Actualmente a principal utilização de árvores Merkle é tornar seguros os blocos de dados recebidos de outros pares nas redes peer-to-peer, para garantir que são recebidos sem danos e sem serem alterados.

É utilizado para verificar que uma fila que tem as transacções a processar não foi modificada e para assegurar a sua integridade para dispersão em todos os nós da rede Mini BlocklyChain. Todos os nós têm de executar este algoritmo para assegurar a integridade de cada transacção que será aplicada.

Bloco para obter um valor da matriz predefinida "Itol_UTXO" (**GetValueArrayItol**).

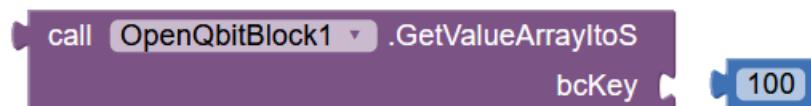


Parâmetros de entrada: **bcKey** <Integer>

Parâmetros de saída: Retorna o valor <Integer> armazenado na etiqueta com o número dado como entrada.

Descrição: É um pedido para a disposição temporária do valor chave, que é predefinido com o nome interno "Itol_UTXO", isto é útil para processar transacções UTXO.

Bloco para obter um valor da matriz predefinida "Itos_UTXO" (**GetValueArrayItos**).



Parâmetros de entrada: **bcKey** <Integer>

Parâmetros de saída: Retorna o valor <Calça> armazenado na etiqueta com o número dado como entrada.

Descrição: É um pedido para a disposição temporária do valor chave, que é predefinido com o nome interno "ItoS_UTXO", isto é útil para processar transacções UTXO.

Bloco para obter um valor da matriz pré-definida "Stol_UTXO" (**GetValueArrayStol**).



Parâmetros de entrada: **bcKey** < String >

Parâmetros de saída: Retorna o valor <Integer> armazenado na etiqueta com o nome dado como entrada.

Descrição: É um pedido para a disposição temporária do valor chave, que é predefinido com o nome interno "StoS_UTXO", isto é útil para processar transacções UTXO.

Bloco para obter um valor da matriz predefinida "StoS_UTXO" (**GetValueArrayStoStoS**).



Parâmetros de entrada: **bcKey** < String >

Parâmetros de saída: Retorna o valor <Calça> armazenado na etiqueta com o nome dado como entrada.

Descrição: É um pedido para a disposição temporária do valor chave, que é predefinido com o nome interno "StoS_UTXO", isto é útil para processar transacções UTXO.

Validador de bloco da cadeia de blocos. (**IsChainValid**)

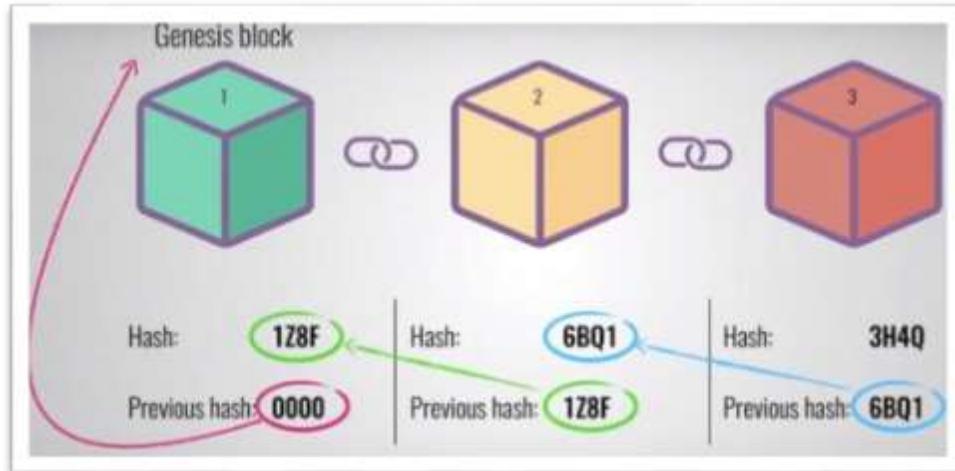


Dependência necessária: Bloco (**GenerateKeyPais**), Bloco (**SenderLoadKeyPair**), Bloco (**RecieipientLoadKeyPair**), Bloco (**GenerateSignature**). Prefácio destes blocos antes da sua utilização.

Parâmetros de entrada: Não aplicável.

Parâmetros de saída: Entrega "Verdadeiro" se a validação da cadeia de blocos estiver correcta ou "Falso" no caso de a validação falhar.

Descrição: Fornece-nos a validação dos componentes que foram previamente inseridos no sistema de cadeia de blocos, esta validação é a mais importante de todo o sistema. Como funciona a validação da cadeia de blocos ou como é geralmente conhecida de uma forma genérica (BlockChain). É a parte central de cada sistema.



Como vemos na imagem anterior, cada bloco que é adicionado no sistema de armazenamento está relacionado com o bloco anterior através dos algoritmos de hash.

Por exemplo, ao criar um novo bloco a acrescentar, o hash que representa a nova informação é obtido tomando o hash do novo bloco de informação + o hash anterior. Com este tipo de ligação será detectada qualquer modificação mínima no armazenamento de cadeias de blocos, o que permite uma segurança de dados muito elevada e eficaz.

Abaixo está um exemplo de como uma cadeia de blocos é armazenada numa base de dados SQLite, veremos como o hash anterior está ligado ao novo hash do último bloco (última fila de transacções processadas) que foi inserido. Cada hash tanto nas colunas "prevhash" como "newhash" representa a informação da fila de transacções que foi processada na altura.

SQLite Expert Personal 5.3 (x64)

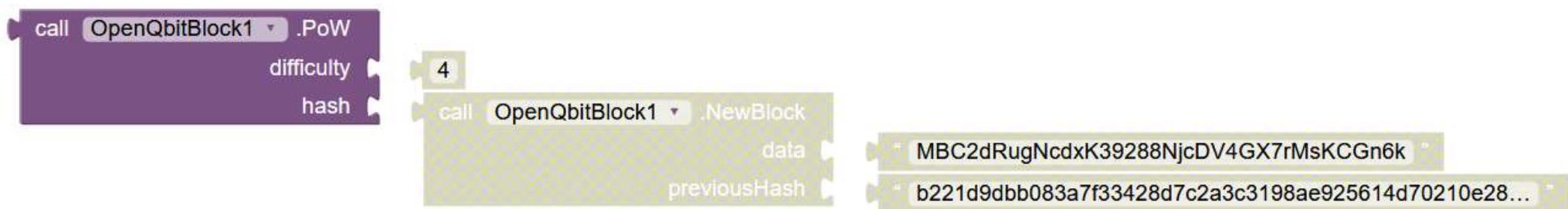
File View Database Object SQL Transaction Tools Help

Database: miniblock Table: nblock File: C:\memo\thunkable\miniblock.db

rowid	id	prevhash	newhash	opera	trans	balan
1	1	10767c864cef0334f27473902eb9868e7	bdc9065d20a4cd037bb1a7538486403e	2	0	0
2	2	bdc9065d20a4cd037bb1a7538486403e	6619f4809d73a267a4b9ac554bb4523a	1	5	5
3	3	6619f4809d73a267a4b9ac554bb4523a	4d186321c1a7f0f354b297e8914ab240	1	5	5

O haxixe anterior está relacionado com o novo haxixe.

Bloco para realizar o Teste de Trabalho de PoW e para extrair novos blocos. (**Pow**)



Dependência obrigatória: **NewBlock**. Prefácio deste bloco antes de o utilizar.

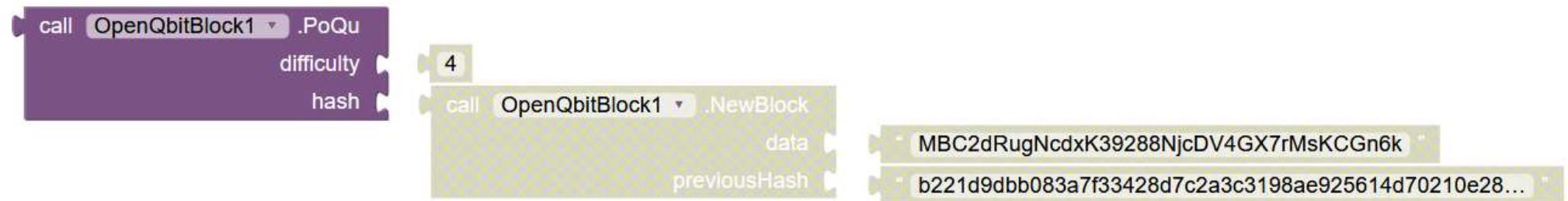
Parâmetros de entrada: **dificuldade** <Integer>, hash <Dependência obrigatória>

Parâmetros de saída: Dá como saída um hash "SHA256" composto com o #Número de "zeros" dado na dificuldade do parâmetro de entrada.

Descrição: Este bloco executa uma actividade chamada "mineração" um novo bloco é o que descrevemos anteriormente como o processo de PoW (Prova de Trabalho), ver secção O que é a Prova de Quantum (PQu)?

Minerar ou executar o PoW é um conceito em que os nós recebem uma tarefa para resolver um puzzle matemático. Quem o resolver primeiro no caso do Mini BlocklyChain tem a oportunidade de continuar com o processo de modo a ser escolhido para ser o vencedor de poder processar a fila de transacção que está à espera de ser processada.

Bloco para realizar o Teste de Trabalho de PoW e para extrair novos blocos. (**PoQu**)



Dependência obrigatória: **NewBlock**. Prefácio deste bloco antes de o utilizar.

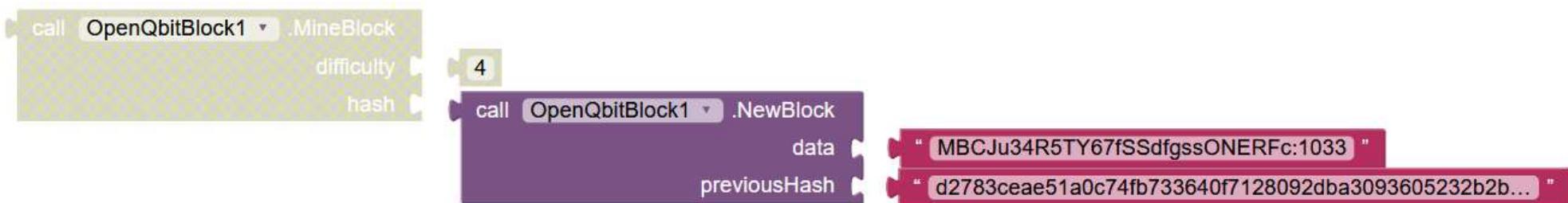
Parâmetros de entrada: **dificuldade** <Integer>, hash <Dependência obrigatória>

Parâmetros de saída: Resultados no número "Número mágico" e número quântico aleatório que está dentro do intervalo de 0 e 1 tipo decimal de 16 dígitos, exemplo (0. 5843012986202495) e um hash "SHA256" composto com o número #Número de "zeros" dado na dificuldade do parâmetro de entrada.

Descrição: Este bloco executa o processo de "mineração" de um novo bloco é o que descrevemos anteriormente como o processo de PoW (Proof of Work), mas este processo chama a função de número quântico aleatório gerado após o cálculo do número "nonce" adequado para satisfazer o nível de dificuldade que pode estar no intervalo de mínimo 1, máximo 5. Ver secção O que é a Prova de Quantum (PQu - Proof Quantum)?

Minerar ou executar o PoW ou PoQu é um conceito em que os nós recebem uma tarefa para resolver um puzzle matemático. O nó que o resolve primeiro no caso de qualquer sistema de cadeia de bloqueio tem a oportunidade de continuar com o processo de modo a ser escolhido para ser o vencedor de poder processar a fila de transacção que está à espera de ser processada.

Bloco para a criação de **novos** blocos (**NewBlock**).

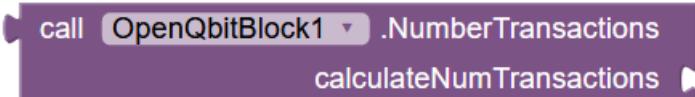


Parâmetros de entrada: **dados** <String>, **previousHash** <String>

Parâmetros de saída: Fornece um hash calculado como: **SHA256 (dados (parâmetros de entrada) + previousHash (parâmetro de entrada) + IMEI (parâmetro interno) + MerkleRoot (parâmetro interno))**.

Descrição: Este bloco realiza a criação de um novo bloco a ser processado. Dá como saída um hash "SHA256" composto pela cadeia de parâmetros de entrada no caso dos dados variar dependendo da concepção de cada sistema e o hash anterior é baseado no hash da cadeia de transacção anterior que já foi processado e é armazenado no sistema de cadeia de blocos Mini BlocklyChain, estes dois são parâmetros variáveis, existem dois parâmetros internos do sistema que são fixos e asseguram a integridade da informação e do sistema, estes dois parâmetros internos são a identificação única do telemóvel e o hash da árvore merklet da fila de transacção que está a ser processada.

Bloco do total das transacções locais do nó (**NumberTransactions**)



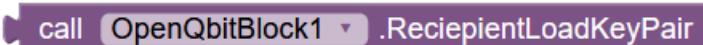
Dependência obrigatória: Block (**AddKeyValueArrayStoS**). Prefácio deste bloco antes de o utilizar.

Parâmetros de entrada: < Dependência obrigatória>.

Parâmetros de saída: Retorna o total de transacções locais ao nó.

Descrição: Fornece o total de transacções que serão aplicadas apenas às contas locais do nó.

Bloco para leitura do endereço do destinatário em ficheiro binário. (**RecieipientLoadKeyValuePair**)

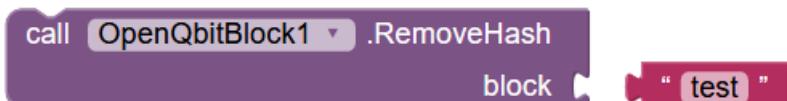


Parâmetros de entrada: **Não aplicável**.

Parâmetros de saída: Retorna chave privada e chave pública em formato Base64.

Descrição: Obtém o endereço privado e público do destinatário a partir de um ficheiro binário como parâmetro de entrada.

Bloco para eliminar elemento em "cadeia" de disposição temporária interna (**RemoveHash**).

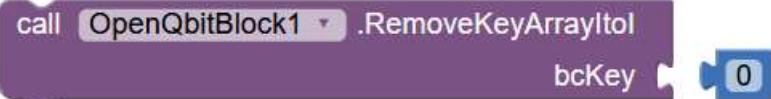


Parâmetros de entrada: **bloco < String>**.

Parâmetros de saída: Não aplicável.

Descrição: Obtém o endereço privado e público do destinatário a partir de um ficheiro binário como parâmetro de entrada.

Bloco para eliminar elemento na disposição temporária interna "Itol_UTXO" (RemoveKeyArrayItol)

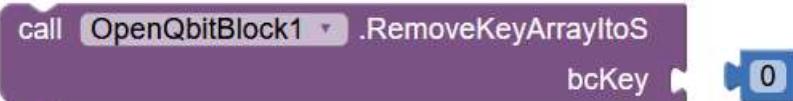


Parâmetros de entrada: **bcKey < Integer >**.

Parâmetros de saída: Não aplicável, tipo de elemento apagar <Integer>

Descrição: O elemento associado ao rótulo numérico da disposição temporária interna "Itol_UTXO" é eliminado.

Bloco para eliminar elemento na disposição temporária interna "ItoS_UTXO" (RemoveKeyArrayItoS).

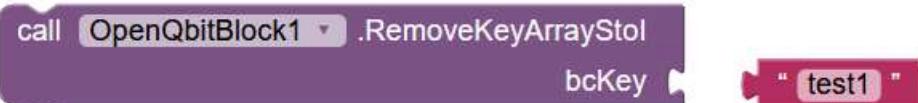


Parâmetros de entrada: **bcKey < Integer >**.

Parâmetros de saída: Não aplicável, tipo de elemento a eliminar <Calça>

Descrição: O elemento associado à etiqueta numérica da disposição temporária interna "ItoS_UTXO" é eliminado.

Bloco para eliminar elemento no arranjo temporário interno "Stol_UTXO" (RemoveKeyArrayStol)

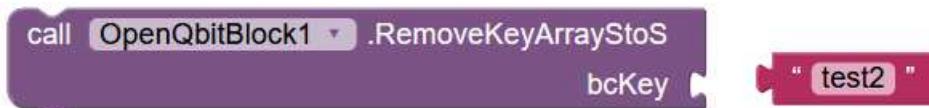


Parâmetros de entrada: **bcKey < String >**.

Parâmetros de saída: Não aplicável, tipo de elemento apagar <Integer>

Descrição: O elemento associado à etiqueta numérica da disposição temporária interna "Stol_UTXO" é eliminado.

Bloco para eliminar elemento em arranjo temporário interno "StoS_UTXO" (**RemoveKeyArrayStoS**)



Parâmetros de entrada: **bcKey < String>**.

Parâmetros de saída: Não aplicável, tipo de elemento a eliminar <Calça>

Descrição: Eliminar elemento da etiqueta de disposição temporária interna "StoS_UTXO".

Bloco para substituir um valor da "corrente" temporária interna (**ReplaceHash**).



Parâmetros de entrada: **bloco < String>, índice < Inteiro>**

Parâmetros de saída: Não aplicável.

Descrição: O elemento associado ao índice "índice" é substituído pelo valor da etiqueta "bloco" na disposição interna temporária "cadeia".

Bloco para iniciar (**enviar**) nova transacção (**SendTrasaction**).



Parâmetros de entrada: **deAddrMBC < String>, toAddrMBC < String>, valor < Inteiro>**

Parâmetros de saída: Retorna o valor "Verdadeiro" se a transacção foi enviada com sucesso ou "Falso" se não foi bem sucedida ele enviou.

Descrição: Bloco que transforma o endereço do remetente e do destinatário num formato binário e é anexado à fila de transacção a ser processada.

Bloco para ler o endereço do remetente em ficheiro binário. (**SenderLoadKeyPair**)

call OpenQbitBlock1 .SenderLoadKeyPair

Parâmetros de entrada: **Não aplicável**.

Parâmetros de saída: Retorna chave privada e chave pública em formato Base64.

Descrição: Obtém o endereço privado e público do remetente a partir de um ficheiro binário como parâmetro de entrada.

Bloco para obter o número do elemento da "cadeia" do acordo temporário (**SizeBlockList**).

call OpenQbitBlock1 .SizeBlockList

Parâmetros de entrada: **Não aplicável**.

Parâmetros de saída: Retorna o número do elemento da "cadeia" da matriz interna.

Descrição: Obtém o número do elemento da "cadeia" da matriz interna.

Bloco para obter o número de elemento da disposição temporária "Itol_UTXO" (**SizeItol**)

call OpenQbitBlock1 .SizeItol

Parâmetros de entrada: **Não aplicável**.

Parâmetros de saída: Retorna o número do elemento da matriz interna "Itol_UTXO".

Descrição: Obtém o número do elemento da matriz interna "Itol_UTXO".

Bloco para obter o número do elemento da disposição temporária "ItoS_UTXO" (**SizeItoS**)

call OpenQbitBlock1 .SizeItoS

Parâmetros de entrada: **Não aplicável**.

Parâmetros de saída: Retorna o número do elemento da matriz interna "ItoS_UTXO".

Descrição: Obtém o número do elemento da matriz interna "ItoS_UTXO".

Bloco para obter o número do elemento da disposição temporária "StoL_UTXO" (**SizeStoL**)



Parâmetros de entrada: **Não aplicável**.

Parâmetros de saída: Retorna o número do elemento da matriz interna "StoL_UTXO".

Descrição: Obtém o número do elemento da matriz interna "StoL_UTXO".

Bloco para obter o número do elemento da disposição temporária "StoS_UTXO" (**SizeStoS**)

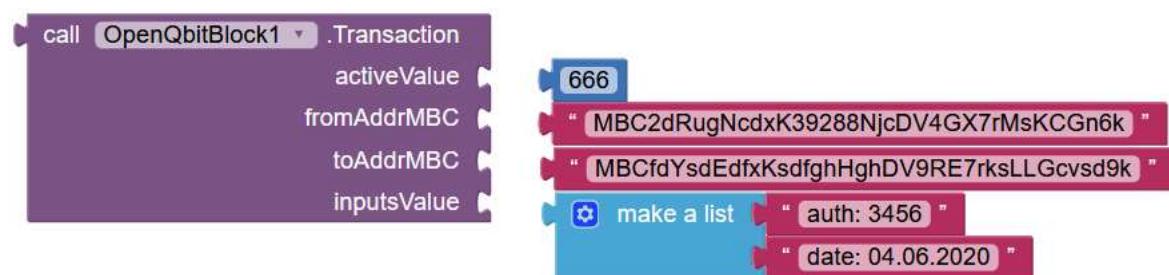


Parâmetros de entrada: **Não aplicável**.

Parâmetros de saída: Retorna o número do elemento da matriz interna "StoS_UTXO".

Descrição: Obtém o número do elemento da matriz interna "StoS_UTXO".

Criação de blocos de transacção com valores adicionais (**Transacção**).



Parâmetros de entrada: **activeValue < Integer>, fromAddrMBC < String>, toAddrMBC < String>, inputValue < Array String>**.

Parâmetros de saída: Dá-nos, os endereços em formato binário e o valor inputstValue convertido numa string 'String', e dá-nos o hash "SHA256" do valor ActiveValue.

Descrição: Prepara uma nova transacção para ser processada pelos nós.

Mini bloco de validação de endereços BlocklyChain (**ValidateAddMiniBlocklyChain**)



Parâmetros de entrada: Endereços de utilizador em formato Mini BlocklyChain.

Parâmetros de saída: Retorna "Verdadeiro" se o endereço estiver no formato correcto ou "Falso" se o endereço for inválido.

Descrição: Valida se o endereço Mini BlocklyChain introduzido está correcto, este bloco aplica um algoritmo para verificar se o endereço foi criado utilizando o mecanismo de criação de endereços a ser utilizado no sistema Mini BlocklyChain.

Bloco de validação de endereços Bitcoin. (**ValidateAddBitcoin**)

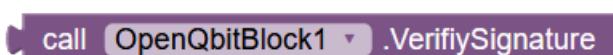


Parâmetros de entrada: **addr** <String>, endereços de utilizador com formato Bitcoin (aceita endereços Bitcoin com identificador inicial "1", endereços com identificador "3" não são aplicáveis).

Parâmetros de saída: Retorna "Verdadeiro" se o endereço estiver no formato correcto ou "Falso" se o endereço for inválido.

Descrição: Valida se o endereço Bitcoin introduzido estiver correcto, este bloco aplica um algoritmo para verificar se o endereço foi criado utilizando o mecanismo de criação de endereço a ser utilizado no sistema Bitcoin.

Bloco de verificação de assinatura digital para a transacção em curso. (**VerifySignature**).



Parâmetros de entrada: **Não aplicável**.

Parâmetros de saída: Retorna "Verdadeiro" se a verificação for válida ou "Falso" se a verificação for inválida.

Descrição: Obtém a verificação da assinatura digital que o remetente deveria ter visto feita no processo de envio da transacção que pretende efectuar. Neste processo de verificação é efectuado para verificar se o valor da fonte enviada não foi alterado no canal para onde a transacção foi enviada, bem como para verificar o destinatário ao qual a transacção deve ser aplicada

20. Utilização de blocos para base de dados SQLite (versão MiniSQLite)

Nesta secção veremos como utilizar os blocos para realizar duas operações principais em que estamos interessados para a funcionalidade do sistema Mini BlocklyChain que é "INSERIR" dados na cadeia de blocos e consultá-los.

NOTA: As transacções na base de dados SQLite são diferentes das transacções enviadas pelos nós a serem aplicadas no sistema Mini Blocklychain.

As transacções na base de dados baseiam-se num modelo CRUD (Create, Read, Update and Delete) e são os processos que só podem ser executados com dados externos ou internos na base de dados SQLite. Na cadeia de blocos são utilizados principalmente os processos de Criação e Leitura, para segurança são descartados os processos de actualização ou eliminação e mais onde é armazenada a cadeia de blocos que dá a segurança integral do sistema.

Por outro lado, as transacções enviadas pelos nós referem-se a todos os processos que envolvem a acção de enviar algum tipo de activo entre os membros (nós) do sistema Mini BlocklyChain, este tipo de transacções inclui diversos processos para a sua aplicação, estes podem ser desde a criação de um endereço digital, uma assinatura digital, uma validação de assinatura, um processo dentro da base de dados SQLite, entre outros processos.

Vamos começar com a definição e utilização dos blocos da base de dados SQLite versão MiniSQLite esta versão está integrada apenas por 8 blocos. Tem uma versão completa para manipular os dados na base de dados SQLite, contudo, para fins práticos, o sistema Mini BlocklyChain com a versão MiniSQLite é suficiente.

No caso de querer rever todos os componentes da sua utilização e descrição, ver Anexo "Blocos alargados para base de dados SQLite".

Blocos de versão MiniSQLite:

Bloqueio para iniciar algum tipo de transacção na base de dados SQLite (**BeginTransaction**)

call OpenQbitQSQLite1 .BeginTransaction

Unidade(s) obrigatória(s): Bloco (**ImportDatabase**), Bloco (**OpenDatabase**).

Parâmetros de entrada: **Utilização antes de < Dependência(s)>** Obrigatória(s)>

Parâmetros de saída: Não aplicável, inicia uma transacção numa base de dados SQLite.

Descrição: bloco que inicia um processo na base de dados SQLite, é necessário ter executado primeiro o bloco de importação da base de dados (**ImportDatabase**) e o bloco aberto da base de dados (**OpenDatabase**).

Bloco para fazer Commit in SQLite. (**CommitTrasaction**)

call **OpenQbitQSQLite1** .**CommitTransaction**

Dependência(ões) necessária(s): Bloco (**BeginTransaction**), Bloco (**ImportDatabase**), Bloco (**OpenDatabase**).

Parâmetros de entrada: **Utilização antes de < Dependência(s)>** Obrigatória(s)>

Parâmetros de saída: Não aplicável, efectua um **commit de uma** transacção numa base de dados SQLite.

Descrição: bloco que inicia um processo de **submissão na** base de dados SQLite, deve primeiro ter executado o bloco de importação da base de dados (**ImportDatabase**) e o bloco aberto da base de dados (**OpenDatabase**).

Bloco para fechar a base de dados SQLite que foi importada ou exportada (**CloseDatabase**)

call **OpenQbitQSQLite1** .**CloseDatabase**

Unidade(s) obrigatória(s): Bloco (**ImportDatabase**), Bloco (**OpenDatabase**).

Parâmetros de entrada: **Utilização antes de < Dependência(s)>** Obrigatória(s)>

Parâmetros de saída: Não aplicável, fecha uma base de dados SQLite.

Descrição: bloco que fecha a base de dados SQLite, é necessário ter executado primeiro o bloco de importação da base de dados (**ImportDatabase**) e o bloco aberto da base de dados (**OpenDatabase**).

Bloco para exportar uma base de dados SQLite (**ExportDatabase**).

call **OpenQbitQSQLite1** .**ExportDatabase**
 fileName **“ mbcExport.sqlite ”**

Unidade(s) obrigatória(s): Bloco (**ImportDatabase**), Bloco (**OpenDatabase**).

Parâmetros de entrada: **fileName < String>** introduzir um caminho onde se pode encontrar uma base de dados já criada com o formato SQLite.

Utilização antes < Dependência(ões) obrigatória(s) >

Parâmetros de saída: Exportar para uma base de dados SQLite.

Descrição: bloco que exporta uma base de dados SQLite, deve ter executado primeiro o bloco de importação da base de dados (**ImportDatabase**) e o bloco aberto da base de dados (**OpenDatabase**).

Bloco para importar base de dados SQLite. (**ImportDatabase**)

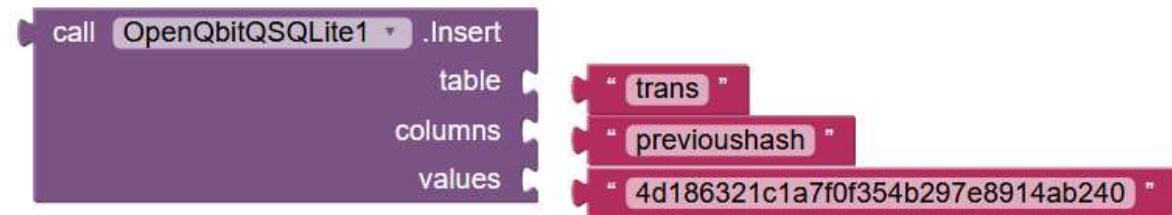


Parâmetros de entrada: **fileName < String >** introduzir um caminho onde se pode encontrar uma base de dados já criada com o formato SQLite.

Parâmetros de saída: Inicia uma base de dados SQLite para executar transacções.

Descrição: bloco que inicia um processo na base de dados SQLite, é necessário ter executado primeiro o bloco de importação da base de dados (**ImportDatabase**) e o bloco aberto da base de dados (**OpenDatabase**).

Bloco para inserir dados na base de dados SQLite. (**Inserir**)



Unidade(s) obrigatória(s): Bloco (**ImportDatabase**), Bloco (**OpenDatabase**).

Parâmetros de entrada: **tabela < String >**, **colums < String >**, **valores < String >**, **Utilização antes < Dependência(ões) > Obrigatória(s)**

Parâmetros de saída: Insere uma transacção numa base de dados SQLite.

Descrição: bloco que insere dados na base de dados SQLite, deve ter executado primeiro o bloco de importação da base de dados (**ImportDatabase**) e o bloco aberto da base de dados (**OpenDatabase**).

Bloco para abrir base de dados SQLite (**OpenDatabase**)



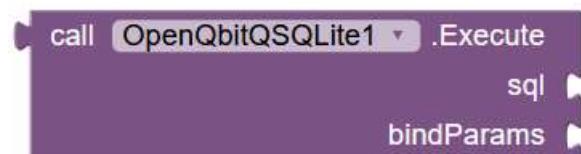
Unidade(s) obrigatória(s): Bloco (**ImportDatabase**).

Parâmetros de entrada: **Utilização antes de** < Dependência(s)> Obrigatória(s)>

Parâmetros de saída: Não aplicável, iniciar ou abrir uma base de dados SQLite para realizar transacções.

Descrição: Bloco que inicia uma base de dados SQLite, tem de haver antes.

Bloco para consulta de dados em SQLite (**Execute**).



Unidade(s) obrigatória(s): Bloco (**ImportDatabase**), Bloco (**OpenDatabase**).

Parâmetros de entrada: **sql** < String> , **bindParams** < String> , **Use antes de** < Dependência(s)> Obrigatória(s)>

Parâmetros de saída: A instrução SQL é executada para criar uma transacção numa base de dados SQLite.

Descrição: bloco que executa instruções SQL na base de dados SQLite, deve primeiro ter executado o bloco de importar base de dados (**ImportDatabase**) e o bloco de abrir a base de dados (**OpenDatabase**).

21. Definição e utilização de blocos de segurança.

Nesta sessão iremos rever a utilização dos blocos que nos fornecem níveis de segurança para guardar, validar e transferir transacções dos nós da rede Mini BlocklyChain.

Os blocos de segurança baseiam-se na seguinte extensão:

- I. OpenQbitAESEncryption extensão.
- II. OpenQbitAESExpansão da descodificação.
- III. Extensão OpenQbitAESToString.
- IV. Extensão OpenQbitEncDecData.
- V. Extensão OpenQbitFileHash.
- VI. Extensão OpenQbitRSA.
- VII. Extensão OpenQbitSSHClient (seria necessário)
- VIII. Extensão OpenQbitStringHash.

Com excepção da extensão OpenQbitSSHClient que é obrigatória, as extensões acima referidas são opcionais para utilização na criação de uma rede pública ou privada de Mini BlocklyChain.

No entanto, a fim de ter um sistema seguro para as suas transacções dependendo de cada caso comercial, a utilização das extensões que são opcionais deve ser aplicada à sua discrição.

Por exemplo, no caso da utilização de hash podemos utilizar uma série de opções de algoritmos, tais como MD5, SHA1, SHA128, SHA256, SHA512 para uma cadeia de caracteres, bem como ser aplicado a qualquer tipo de ficheiro, dependendo do fluxo de informação de cada sistema criado com Mini BlocklyChain.

OpenQbitAESEncryption extensão.

Bloco para encriptar ficheiro com segurança AES. (**AESEncryption**).



Parâmetros de entrada: **pathFileIn < String>** , **pathFileOut < String>** , **pathFile < String>**, **pathFileVI < String>** e **passwd < String>**. Os nomes dos ficheiros são arbitrários e à disposição de cada concepção de sistema.

Parâmetros de saída: ficheiro encriptado AES introduzido no **caminho do parâmetro de entradaFileIn**.

Descrição: Bloco que nos dá três ficheiros de saída, um deles é o ficheiro de origem já encriptado pelo algoritmo AES com uma chave de 256 bits, os outros dois ficheiros são utilizados para controlar a extensão para decifrar e recuperar o ficheiro original.

OpenQbitAESExpansão da descodificação.

Bloco para decifrar ficheiro AES. (**AESDecryption**).



Parâmetros de entrada: **pathFileIn < String>** , **pathFileOut < String>** , **pathFile < String>**, **pathFileVI < String>** e **passwd < String>**. Os nomes dos ficheiros são os mesmos que os obtidos como resultado no bloco (**AESEncryption**).

Parâmetros de saída: Ficheiro original desencriptado com AES introduzido no parâmetro de **entradaFileIn**, neste caso para desencriptar o ficheiro é introduzido no **pathFileIn** no ficheiro encriptado e no **pathFileOut** dar-nos-á o ficheiro original (desencriptado).

Descrição: Bloco que nos dá um ficheiro no parâmetro PathFileOut que será a saída desencriptada pelo algoritmo AES com uma chave de 256 bits.

Extensão OpenQbitAESToString.

Esta extensão é de utilização única por sessão de dispositivo, ou seja, só funciona quando o dispositivo não é reinicializado (telemóvel), uma vez que o valor da encriptação VI é gerado temporariamente.

NOTA: Se o dispositivo for reinicializado, a cadeia encriptada não pode ser recuperada.

Bloco para encriptação temporária de cadeia de caracteres (**DecrypSecretText**)

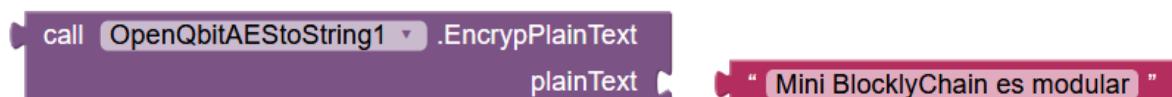


Parâmetros de entrada: **secretText** < String >

Parâmetros de saída: cadeia de caracteres original desencriptada com AES com chave de 256 bits

Descrição: Bloco que nos dá uma sequência de caracteres, é o parâmetro de entrada que foi introduzido no bloco (**EncrypPlainText**).

Bloco para descodificar um fio (**EncrypPlainText**)



Parâmetros de entrada: **plainText** < String >

Parâmetros de saída: Cadeia de caracteres encriptada AES usando chave de 256 bits.

Descrição: Bloco que nos dá uma cadeia de caracteres alfanuméricos encriptados com AES usando uma chave digital de 256 bits.

Extensão OpenQbitEncDecData.

Bloco de encriptação especializado para bases de dados genéricas (**Dados criptográficos**)



Parâmetros de entrada: **plainText** < String >

Parâmetros de saída: Cadeia de caracteres encriptada AES usando chave de 256 bits.

Descrição: Bloco que nos dá uma cadeia de caracteres alfanuméricos encriptados com AES usando uma chave digital de 256 bits.

Bloco de encriptação especializado para bases de dados genéricas (**DescriptionData**)



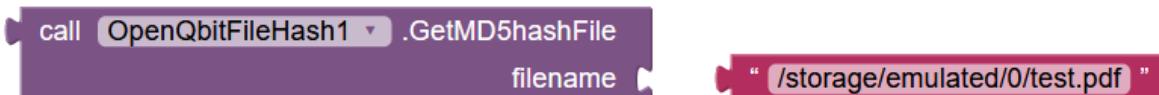
Parâmetros de entrada: **plainText** < String >

Parâmetros de saída: Cadeia de caracteres encriptada AES usando chave de 256 bits.

Descrição: Bloco que nos dá uma cadeia de caracteres alfanuméricos encriptados com AES usando uma chave digital de 256 bits.

Extensão OpenQbitFileHash.

Bloco para gerar MD5 a partir de um ficheiro (**GetMD5hashFile**)



Parâmetros de entrada: **nome de ficheiro** <string>

Parâmetros de saída: Fornece o ficheiro hash MD5.

Descrição: Bloco para criar o hash MD5 do ficheiro dado no parâmetro de entrada.

Bloco para gerar SHA256 a partir de um ficheiro (**GetSHA256hashFile**)



Parâmetros de entrada: **nome de ficheiro** <string>

Parâmetros de saída: Fornece o hash de arquivo SHA256.

Descrição: Bloco para criar o hash SHA256 do ficheiro dado no parâmetro de entrada.

Bloco para gerar SHA512 a partir de um ficheiro (**GetSHA512hashFile**)

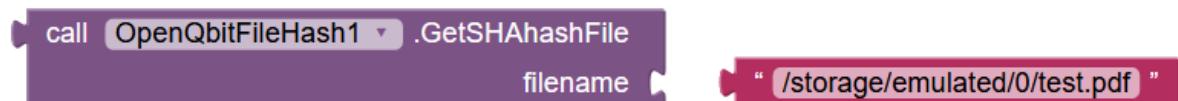


Parâmetros de entrada: **nome de ficheiro** <string>

Parâmetros de saída: Fornece o hash de arquivo SHA256.

Descrição: Bloco para criar o hash SHA256 do ficheiro dado no parâmetro de entrada.

Bloco para gerar SHA1 a partir de um ficheiro (**GetSHA1hashFile**)



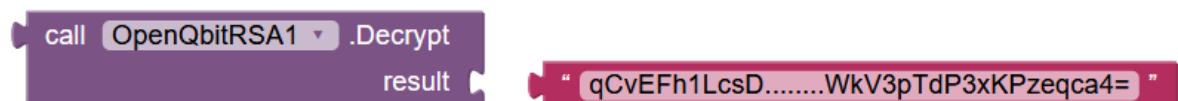
Parâmetros de entrada: **nome de ficheiro** <string>

Parâmetros de saída: Fornece o hash de arquivo SHA1.

Descrição: Bloco para criar o hash SHA1 do coto no parâmetro de entrada.

Extensão OpenQbitRSA.

Bloco para **decifrar** cordel com RSA (**Decrypt**)



Dependência(s) necessária(s): Block (**Encrypt**), Block (**OpenFromDiskPrivateKey**), Block (**OpenFromDiskPublicKey**).

Parâmetros de entrada: **resultado** < String >

Parâmetros de saída: Cadeia de caracteres descodificada com RSA.

Descrição: Bloco que nos dá uma sequência de caracteres alfanuméricos decifrados utilizando uma chave do tamanho que foi utilizada no bloco (**GenKeyPair**).

Bloco de encriptação de cordas com RSA (**Encrypt**)



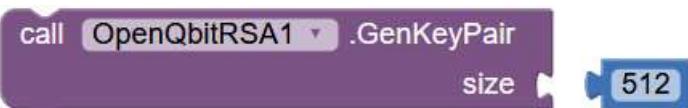
Unidade(s) necessária(s): Block (**GenKeyValuePair**), Block (**SaveFromDiskPrivateKey**), Block (**SaveFromDiskPublicKey**)

Parâmetros de entrada: **simples** < String >

Parâmetros de saída: cadeia de caracteres encriptada por RSA

Descrição: Bloco que nos dá uma sequência de caracteres alfanuméricos decifrados utilizando uma chave do tamanho que foi utilizada no bloco (**GenKeyValuePair**).

Bloco para **decifrar** cordel com RSA (**Decrypt**)

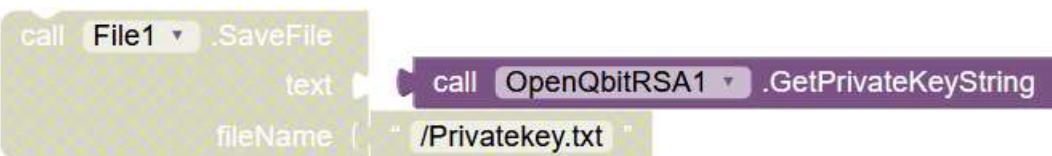


Parâmetros de entrada: **tamanho** < Inteiro >

Parâmetros de saída: Não aplicável.

Descrição: Bloco para gerar chave privada e chave pública com base no tamanho escolhido.

Bloco para obter a chave privada (**GetPrivateKeyString**)



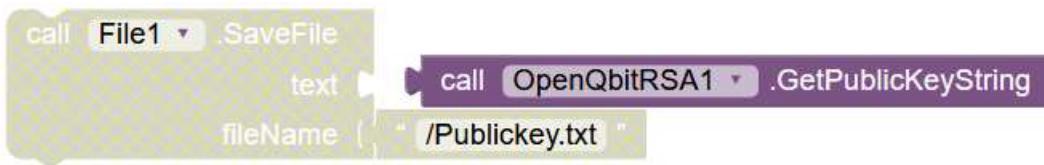
Unidade(s) Obrigatória(s): Bloco (**GenKeyValuePair**), Bloco (**GenKeyValuePair**), Bloco (**Ficheiro**)

Parâmetros de entrada: **Não aplicável**.

Parâmetros de saída: ficheiro com cadeia de caracteres encriptada RSA (chave privada)

Descrição: Bloco que nos dá uma cadeia alfanumérica representando a chave privada encriptada utilizando a chave de tamanho que foi utilizada no bloco (**GenKeyValuePair**).

Bloco para obter a chave privada (**GetPublicKeyString**)



Unidade(s) Obrigatória(s): Bloco (**GenKeyPair**), Bloco (**GenKeyPair**), Bloco (**Ficheiro**)

Parâmetros de entrada: **Não aplicável.**

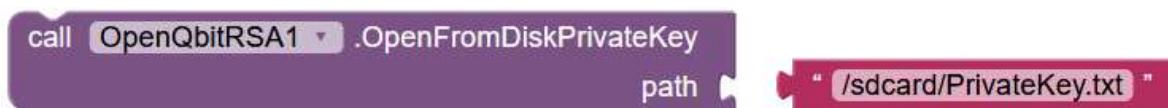
Parâmetros de saída: ficheiro com string encriptada RSA (chave pública)

Descrição: Bloco que nos dá uma cadeia alfanumérica representando a chave pública encriptada utilizando o tamanho da chave utilizada no bloco (**GenKeyPair**).

NOTA: Nos blocos anteriores (**GetPrivateKeyString**) e (**GetPublicKeyString**) nas dependências utilizaremos o bloco genérico (**Ficheiro**) da aplicação App Inventor da sessão "Armazenamento" de paletes.

Esta forma de armazenar a chave privada e pública por meio do bloco (**ficheiro**) pode ajudar-nos a ter uma melhor manipulação da informação.

Bloco para ler chave privada de um ficheiro (**OpenFromDiskPrivateKey**).



Dependência(s) necessária(s): Bloco (**SaveFromDiskPrivateKey**) ou Bloco (**GetPrivateKeyString**).

Parâmetros de entrada: **caminho < String>**

Parâmetros de saída: carga do sistema RSA chave privada encriptada cadeia de caracteres.

Descrição: Bloco que nos dá uma cadeia de caracteres alfanuméricos encriptados da chave privada armazenada no caminho do ficheiro fornecido.

Bloco para ler a chave pública de um ficheiro (**OpenFromDiskPublicKey**)



Unidade(s) Obrigatória(s): Bloco (**SaveFromDiskPublicKey**) ou Bloco (**GetPublicKeyString**).

Parâmetros de entrada: **caminho** < String >

Parâmetros de saída: Carregar no sistema RSA cadeia de caracteres de chave pública encriptada.

Descrição: Bloco que nos dá uma cadeia alfanumérica encriptada da chave pública armazenada no caminho do ficheiro fornecido.

Bloco para guardar a chave privada de um ficheiro (**SaveToDiskPrivateKey**).



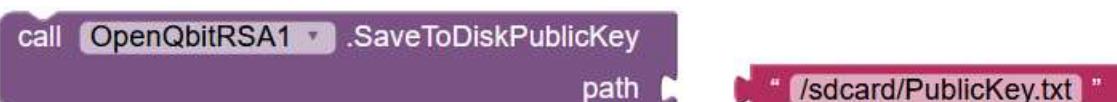
Unidade(s) obrigatória(s): Bloco (**GenKeyValuePair**).

Parâmetros de entrada: **simples** < String >

Parâmetros de saída: ficheiro com string encriptada RSA. (chave privada)

Descrição: Bloco que nos dá um ficheiro com uma cadeia alfanumérica encriptada utilizando uma chave privada do tamanho utilizado no bloco (**GenKeyValuePair**).

Bloco para guardar a chave privada de um ficheiro (**SaveToDiskPublicKey**).



Unidade(s) obrigatória(s): Bloco (**GenKeyValuePair**).

Parâmetros de entrada: **simples** < String >

Parâmetros de saída: ficheiro com string encriptada RSA. (chave pública)

Descrição: Bloco que nos dá um ficheiro com uma cadeia alfanumérica encriptada utilizando uma chave pública do tamanho que foi utilizado no bloco (**GenKeyValuePair**).

Extensão OpenQbitSSHClient.

Bloco **conector** SSH Cliente (**ConnectorMiniBlocklyChain**)



Parâmetros de entrada: **username <string>**, **password <string>**, **host <string>**, **port<integer>**

Parâmetros de saída: Se a ligação com o servidor ssh do terminal Termux for bem sucedida, dá-nos uma mensagem; "**Connect SSH**", se não for bem sucedida, dá-nos uma mensagem **NULL**.

Descrição: Bloco de comunicação para ligar o Mini BlocklyChain ao terminal Termux, através do protocolo de comunicação SSH (Secure Shell).

Bloco de Execução de Comandos no Terminal Linux Termux



(**CommandLineMiniBlocklyChain**).

Parâmetros de entrada: **comando <string>**

Parâmetros de saída: Dados variáveis, dependendo do comando ou programa executado.

Descrição: bloco de execução de comandos no terminal Termux pré-requisito para fazer uma ligação com o bloco (**ConnectorMiniBlocklyChain**) pode executar todos os tipos de comandos online e/ou obter dados de execução específicos a partir de scripts ou programas que tenham uma CLI (Command-Line Interface) online.

DisconnectMiniBlocklyChainSSH.

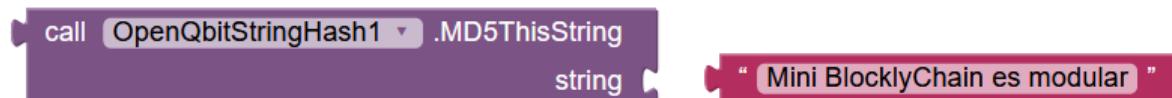


Parâmetros de entrada e saída: Não aplicável (nenhum)

Descrição: Bloco para fechar sessão SSH.

Extensão OpenQbitStringHash.

Bloco para gerar fio MD5 (**MD5ThisString**)

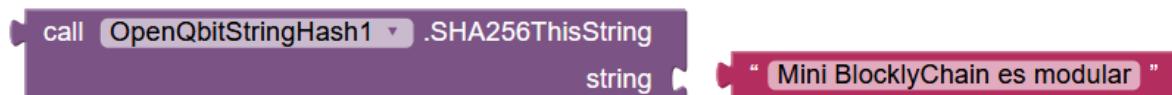


Parâmetros de entrada: string

Parâmetros de saída: Fornece o hash MD5.

Descrição: Bloco para criar o hash MD5 da cadeia dada no parâmetro de entrada.

Bloco para gerar cadeia de caracteres SHA256 (**SHA256ThisString**)

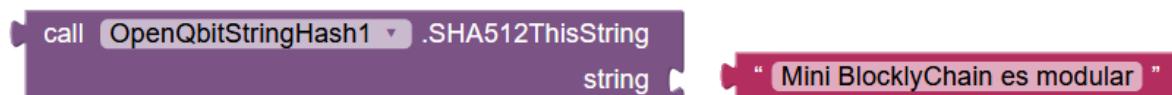


Parâmetros de entrada: string

Parâmetros de saída: Fornece o hash SHA256.

Descrição: Bloco para criar o hash SHA256 da cadeia dada no parâmetro de entrada.

Bloco para gerar fio SHA512 (**SHA512ThisString**)

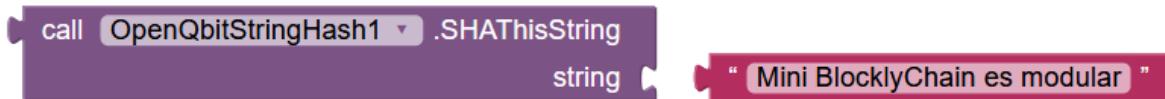


Parâmetros de entrada: string

Parâmetros de saída: Fornece o hash SHA512.

Descrição: Bloco para criar o hash SHA512 da cadeia dada no parâmetro de entrada.

Bloco para gerar cadeia de caracteres SHA1 (**SHAThisString**)



Parâmetros de entrada: string

Parâmetros de saída: Fornece o hash SHA1.

Descrição: Bloco para criar o hash SHA1 da cadeia dada no parâmetro de entrada.

22. Definição de parâmetros de segurança em Mini BlocklyChain.

Os parâmetros de segurança estão divididos em três componentes de qualquer sistema que seja concebido e aplicado nos seguintes componentes:

- a. Base de dados Redis (rede de comunicação de reserva)
- b. Sistema de Sincronização Peer to Peer Syncthing
- c. Integrar a segurança para proteger a base de dados SQLite
- d. Ambiente de desenvolvimento para integração de módulos

a. Base de dados Redis (rede de comunicação de reserva)

Renomeando comandos perigosos, a característica adicional de segurança incorporada da Redis envolve a renomeação ou desactivação de alguns comandos que são considerados perigosos.

Quando executados por utilizadores não autorizados, estes comandos podem ser utilizados para reconfigurar, destruir ou apagar os seus dados. Tal como a senha de autenticação, a renomeação ou desactivação de comandos é configurada na mesma secção de SEGURANÇA do ficheiro `/etc/redis/redis.conf`.

Alguns dos comandos considerados perigosos: **FLUSHDB**, **FLUSHALL**, **KEYS**, **PEXPIRE**, **DEL**, **CONFIG**, **SHUTDOWN**, **BGWRITEAO**, **BGSAVE**, **SAVE**, **SPOP**, **SREM**, **RENAME**, e **DEBUG**. Esta não é uma lista completa, mas renomear ou desactivar todos os comandos dessa lista é um bom começo para melhorar a segurança do seu servidor Redis.

Dependendo das suas necessidades específicas ou das do seu sítio, deve renomear ou desactivar um comando. Se sabe que nunca usará um comando que possa ser manipulado, pode desactivá-lo. Por outro lado, pode querer dar-lhe um novo nome.

Para activar ou desactivar os comandos Redis, reabrir o ficheiro de configuração:

```
$ vi /etc/redis/redis.conf
```

Aviso: Os seguintes passos para desactivar e renomear os comandos são exemplos. Deverá apenas optar por desactivar ou renomear os comandos que se aplicam a si. Pode rever a lista completa de comandos e determinar como eles podem ser mal utilizados no redis.io/commands.

Para desactivar um comando, basta dar-lhe um novo nome para que se torne uma cadeia vazia (simbolizada por um par de aspas sem caracteres entre elas), como se mostra abaixo:

`/etc/redis/redis.conf`

```
.
.
.
# Também é possível matar completamente um comando, renomeando-o para
# um fio vazio:
#
renomear o comando FLUSHDB "".
renomear o comando FLUSHALL "".
renomear DEBUG ""
.
.
```

Para renomear um comando, dar-lhe outro nome, como se mostra nos exemplos abaixo. Os comandos renomeados devem ser difíceis de adivinhar para os outros, mas fáceis de lembrar.

/etc/redis/redis.conf

```
.
.
.
# renomear comando CONFIG ""
renomear comando SHUTDOWN SHUTDOWN_MENOT
renomear comando CONFIG ASC12_CONFIG
```

Guardar as alterações e encerrar o ficheiro.

Após renomear um comando, aplicar a alteração reiniciando a Redis:

- sudo systemctl restart redis.service

Para testar o novo comando, introduza a linha de comando Redis:

- redis-cli

Em seguida, realizar a autenticação:

- auth your_redis_password

Saída
OK

Como no exemplo anterior, suponha que renomeou o comando CONFIG para ASC12_CONFIG. Primeiro, tente usar o comando original CONFIG. Porque mudou o seu nome, não deve funcionar:

- config get requirepass

Saída
(erro) ERR comando desconhecido 'config'

No entanto, o comando renomeado pode ser chamado com sucesso. Não é sensível a maiúsculas e minúsculas:

- asc12_config get requirepass

Saída
1) "requirepass" (passagem obrigatória)
2) "as suas_senha_redis_password"

Finalmente, poderá encerrar o redistritamento:

- saída

Note que se já utilizar a linha de comando Redis e reiniciar a Redis, terá de autenticar novamente. Caso contrário, se escrever um comando, este erro será exibido:

Saída
NOAUTH Autenticação necessária.

b. Sistema de Sincronização Peer to Peer SyncThing

Na utilização do sistema "Peer to Peer" entre nós, o sistema tem os três elementos seguintes aplicados na rede de comunicação

-Privado: não há informação armazenada em nenhum outro lugar para além dos seus computadores. Não existe um servidor central que possa ser comprometido (legal ou ilegalmente).

-Criptografada: Todas as comunicações são protegidas através do protocolo TLS (Transport Layer Security); um protocolo criptográfico que inclui uma sequência perfeita para impedir qualquer pessoa fora da sua confiança de aceder à sua informação.

-Autenticado: Cada nó é identificado com um forte certificado criptográfico. Somente os nós que tenha explicitamente permitido, podem ligar-se à sua informação.

<https://docs.syncthing.net/users/security.html>

Usando o comando online SyncThingManager:

<https://github.com/classicsc/syncthingmanager>

c. Integrar a segurança para proteger a base de dados SQLite

Ao utilizar a extensão (**OpenQbitSQLite**) ou no seu caso a extensão (**ConnectorSSHClient**) para utilizar a CLI SQLite ambas as extensões podem ser combinadas com a extensão de segurança AES (**OpenQbitEncDecData**).

Ver Apêndice "Exemplo de criação do sistema Mini BlocklyChain".

d. Ambiente de desenvolvimento para integração de módulos

A implementação da segurança no ambiente de desenvolvimento pode ser feita através de dois processos:

- Restrição do uso das bibliotecas OpenJDK.
- Utilização restrita a nós autorizados do sistema.

23. Anexo "Criação de bases de dados KeyStore & PublicKeys".

Uma KeyStore é um repositório de certificados de segurança, seja certificados de autorização ou certificados de chave pública, senhas ou chaves de segurança genéricas, tais como as correspondentes chaves privadas (endereços) de um utilizador, que é utilizado para criar ou processar transacções.

É uma base de dados encriptada para que apenas o proprietário possa fazer uso dela. Normalmente é um tipo local, contudo, no sistema Mini BloclyChain é uma base de dados segura mas é partilhada e distribuída em todos os nós, isto deve-se basicamente a uma razão simples, todos os nós devem conhecer os endereços de todos os utilizadores (endereços públicos), os endereços privados são sempre locais e são de uso único e exclusivo de cada utilizador, contudo, ao fazer uma transacção de entrada(depósito) ou saída(despesa) cada transacção tem sempre pelo menos três componentes ao ser criada: endereço de origem, endereço de destino e activo ou valor que está a ser enviado.

Para criar a nossa KeyStore, teremos de seguir os seguintes passos e requisitos.

Um primeiro requisito é ter um tipo de armazenamento onde serão armazenados, no nosso caso utilizaremos a base de dados SQLite e criaremos duas KeyStore uma com as chaves privadas do utilizador que serão locais e será segura "encriptada" a informação e outra base de dados que armazenará as chaves públicas esta base será partilhada em todos os nós da rede, a base que será partilhada na rede será também encriptada, no entanto esta terá uma password que só os membros (nós) da rede serão capazes de partilhar.

O segundo requisito fundamental é o processo de encriptação de informação, isto será feito com a extensão especializada para utilização em base de dados genérica chamada (**OpenQbitEncDecData**) que utiliza um algoritmo AES.

A extensão (**OpenQbitEncDecData**) é funcional para a verificação de elementos unitários da cadeia de blocos, contudo, no caso de ter de verificar a integridade total da cadeia de blocos, não será eficiente, pelo que também estaremos a realizar uma encriptação de uma cópia, mas neste caso será através das extensões: (**OpenQbitAESEncryption**) e (**OpenQbitAESDecryption**) que funcionam num ficheiro, não por dados referenciados.

NOTA: O servidor SSH deve estar a funcionar no terminal Termux para que a base de dados da **KeyStore** funcione correctamente. Executar o comando no terminal:

\$ sshd

Extensões baseadas no algoritmo AES podem ser usadas para qualquer tipo de dados ou ficheiro e este algoritmo foi escolhido por ser o único que provou ser a prova contra ataques baseados em computação quântica.

Criptosistema	Categoría	Tamaño de clave	Parámetro de seguridad	Algoritmo cuántico estimado que rompa el criptosistema	Nº de qubits lógicos necesarios	Nº de qubits físicos necesarios	Tiempo necesario para romper el sistema	Estrategias de reemplazo cuántico-resilientes
AES-GCM	Cifrado simétrico	128	128	Algoritmo de Grover	2.953	$4,61 \times 10^6$	$2,61 \times 10^{12}$ años	
		192	192		4.449	$1,68 \times 10^7$	$1,97 \times 10^{22}$ años	
		256	256		6.681	$3,36 \times 10^7$	$2,29 \times 10^{32}$ años	
RSA	Cifrado asimétrico	1024	80	Algoritmo de Shor	2.290	$2,56 \times 10^6$	3,58 horas	Migrar a un algoritmo PQC seleccionado por el NIST
		2048	112		4.338	$6,2 \times 10^6$	28,63 horas	
		4096	128		8.434	$1,47 \times 10^7$	229 horas	
ECC Problema del logaritmo discreto	Cifrado asimétrico	256	128	Algoritmo de Shor	2.330	$3,21 \times 10^6$	10,5 horas	Migrar a un algoritmo PQC seleccionado por el NIST
		386	192		3.484	$5,01 \times 10^6$	37,67 horas	
		512	256		4.719	$7,81 \times 10^6$	95 horas	
SHA256	Minedo de Bitcoin	N/A	72	Algoritmo de Grover	2.403	$2,23 \times 10^6$	$1,8 \times 10^6$ años	
PBKDF2 con 10.000 iteraciones	Hashing de contraseñas	N/A	66	Algoritmo de Grover	2.403	$2,23 \times 10^6$	$2,3 \times 10^7$ años	Abandonar la autenticación basada en contraseñas

O quadro acima é referido às Academias Nacionais de Ciência, Engenharia e Medicina.

<https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects>

Descrição

A mecânica quântica, o subcampo da física que descreve o comportamento de partículas muito pequenas (quarks), fornece a base para um novo paradigma de computação. Proposta pela primeira vez nos anos 80 como forma de melhorar a modelação computacional de sistemas quânticos, o campo da computação quântica atraiu recentemente uma atenção significativa devido aos progressos na construção de dispositivos de pequena escala. No entanto, serão necessários avanços técnicos significativos antes que um computador quântico prático possa ser alcançado em grande escala.

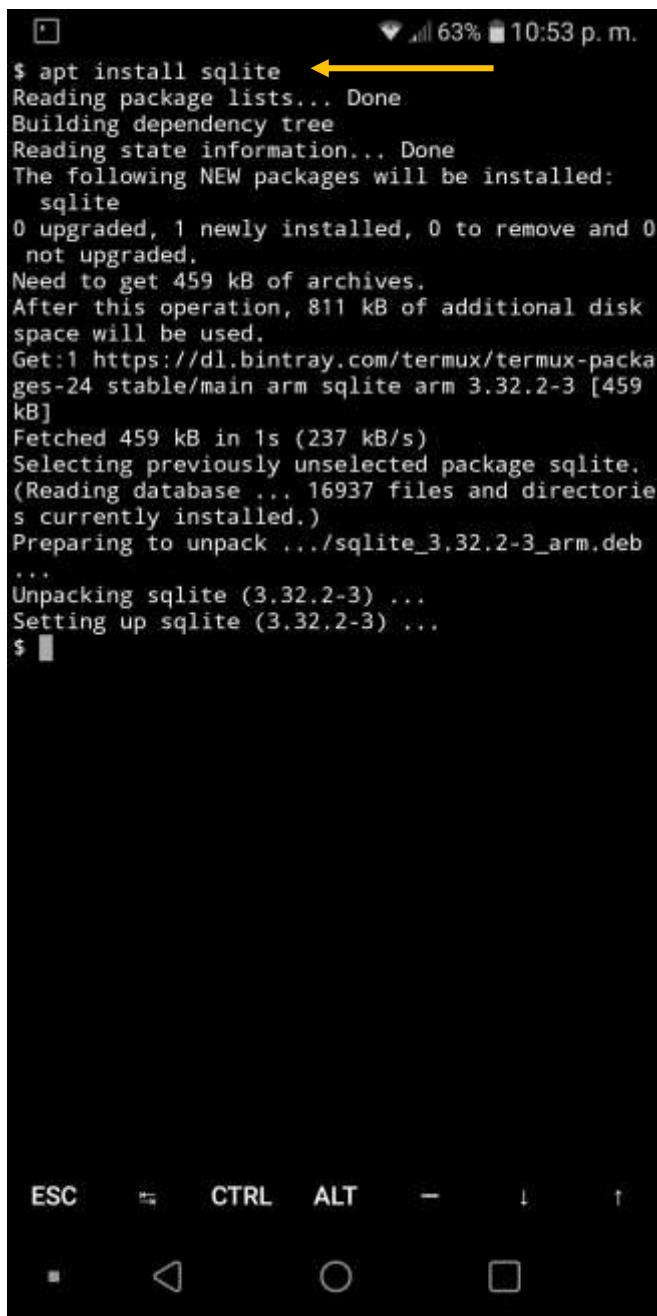
Quantum Computing: Progress and Prospects oferece uma introdução ao campo, incluindo as características únicas e limitações da tecnologia, e avalia a viabilidade e implicações da criação de um computador quântico funcional capaz de lidar com problemas do mundo real. Este relatório considera os requisitos de hardware e software, algoritmos quânticos, controladores de avanços na computação quântica e dispositivos quânticos, parâmetros de referência associados a casos de utilização relevantes, tempo e recursos necessários, e como avaliar a probabilidade de sucesso.

Instalação do manipulador de base de dados SQLite no terminal TERMUX e teste o CLI (Command-Line) do comando **sqlite3** criando uma base de dados chamada **keystore.db**

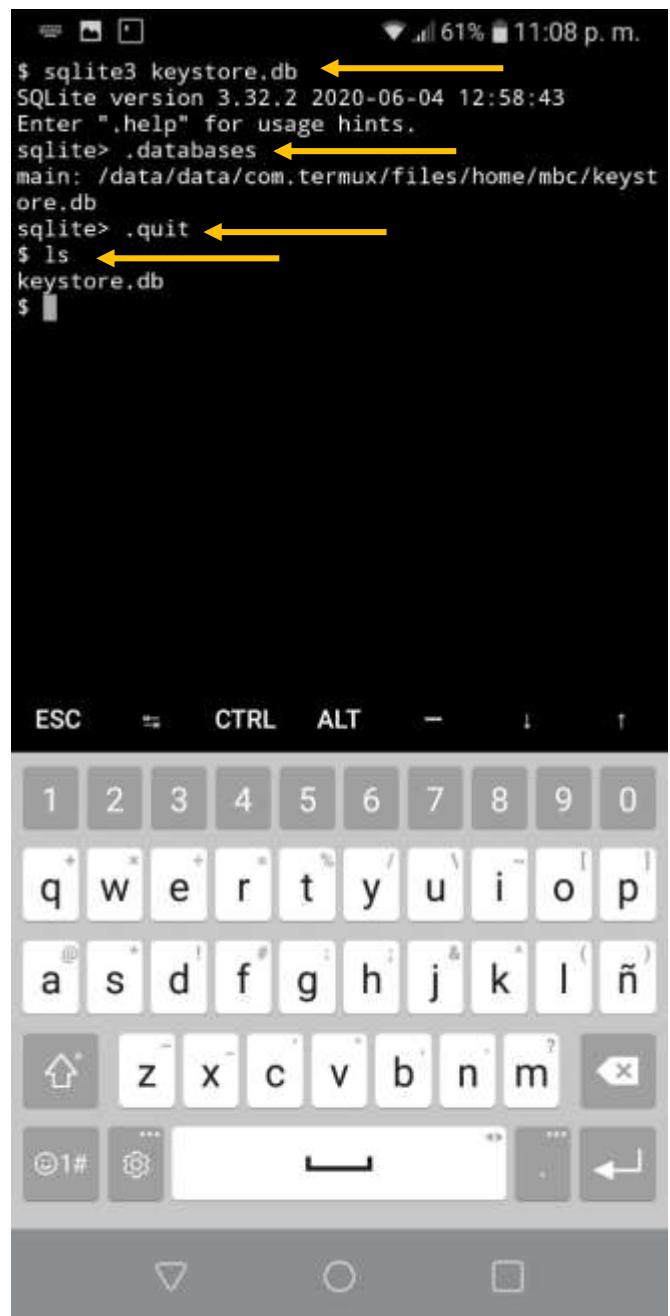
Utilizamos os comandos:

```
$ apt install sqlite
```

```
$ sqlite3 keystore.db
```



```
$ apt install sqlite
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  sqlite
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 459 kB of archives.
After this operation, 811 kB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-packages-24 stable/main arm sqlite arm 3.32.2-3 [459
kB]
Fetched 459 kB in 1s (237 kB/s)
Selecting previously unselected package sqlite.
(Reading database ... 16937 files and directo
ries currently installed.)
Preparing to unpack .../sqlite_3.32.2-3_arm.deb
...
Unpacking sqlite (3.32.2-3) ...
Setting up sqlite (3.32.2-3) ...
$
```



```
$ sqlite3 keystore.db
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
sqlite> .databases
main: /data/data/com.termux/files/home/mcb/keyst
ore.db
sqlite> .quit
$ ls
keystore.db
$
```

Depois dentro do **sqlite> handler** executamos a frase **.bases de dados** para verificar em que caminho está a base de dados que estamos a criar, depois para sair e guardar a base de dados damos a frase de **. desistir**.

NOTA: Em ambas as declarações ou comandos dentro do **sqlite> handler** deve primeiro colocar um ponto "..." na sintaxe.

Finalmente verificamos que a base de dados (vazia) já foi criada, dando o comando:

\$ ls

Procedemos à criação de uma tabela onde as chaves primárias serão armazenadas em três formatos diferentes: hexadecimal, binário e o Mini BlocklyChain endereço de referência do utilizador, que é a chave pública da sua respectiva chave primária.

A encriptação de dados AES será aplicada apenas nos formatos hexadecimal e binário. No caso do endereço do utilizador destinatárioMBC e o pseudónimo não porque é a chave pública que pode e deve ser partilhada através da rede para receber transacções, bem como o pseudónimo para efectuar a pesquisa por este campo.

Criou uma tabela chamada "privatekey" na base de dados em SQLite (keystore.db).

```
CREATE TABLE privatekey (
    id integer chave primária
    t.c.p.          VARCHAR(50) NÃO NULO
    addrHexVARCHAR (65) NOT NULL
    addrMBCVARCHAR (65) NOT NULL
    addrBinBLOB   NÃO NULO
);
```

Vamos executar as frases no CLI sqlite para criar a base de dados keystore.db.

Vamos utilizar novamente a linha de comando sqlite3 com o seguinte comando:

\$ sqlite3

Isto enviar-nos-á dentro do **sqlite>** gestor de **base de dados**. Neste primeiro abriremos a base de dados já criada para poder trabalhar nela com a seguinte frase dentro do gestor:

Sqlite> .open keystore.db

Depois introduziremos a instrução SQL "**CREATE TABLE**" para criar a tabela de **chave privada**.

A seguir, são mostradas duas opções para criar a mesma tabela, uma é através de uma única linha onde estão incluídas todas as instruções SQL dos elementos que a integram. O segundo

exemplo é através da introdução de cada elemento que integra a estrutura de uma forma segmentada.

\$ sqlite3

SQLite versão 3.32.2 2020-06-20 15:25:24

Introduzir ".help" para dicas de utilização.

Ligado a uma base de dados in-memory transitória.

Utilizar ".open FILENAME" para reabrir numa base de dados persistente.

sqlite> .open keystore.db

sqlite> CREATE TABLE privatekey (id integer primary key AUTOINCREMENT NOT NULL, aka VARCHAR(50) NOT NULL, addrHex VARCHAR(65) NOT NULL, addrMBC VARCHAR(65) NOT NULL, addrBin BLOB NOT NULL);

sqlite> .quit

A criação da mesma tabela de **chave privada** é mostrada a seguir, mas é através da introdução da instrução SQL de uma forma segmentada:

```
sqlite> CREATE TABLE privatekey (
...> id integer chave primária AUTOINCREMENTO
...> aka  VARCHAR(50) NOT NULL,
...> addrHex VARCHAR (65) NOT NULL,
...> addrMBC VARCHAR (65) NOT NULL,
...> addrBin BLOBIN BLOB NOT NULL
...> );
sqlite> .tabelas
chave privada
sqlite> .quit
```

Os dois exemplos acima dão o mesmo resultado. Mais tarde executamos a frase **.tables** para verificar se a tabela de chaves privadas foi criada, no final daremos a frase **.quit** com este processo já criámos a tabela de **chaves privadas** dentro da base de dados SQLite **keystore.db**.

Até este momento já temos a estrutura da base de dados **keystore.db** e a sua tabela de chaves privadas onde as chaves privadas serão armazenadas de forma encriptada.

Isto deve mostrar algo semelhante no nó (telemóvel) com o terminal TERMUX.



The screenshot shows a Termux terminal window with a black background. At the top, it displays the command '\$ sqlite3' followed by the SQLite version and usage hints. Below this, the user runs 'CREATE TABLE privatekey (...)' to create a table named 'privatekey'. After creating the table, the user runs '.tables' to list the tables, which shows 'privatekey'. Finally, the user runs '.quit' to exit the SQLite shell. A yellow bracket on the right side of the terminal window groups the first four lines of text (the creation of the table, the listing of tables, and the exit command) under the heading 'Instruções SQL para criar uma nova tabela'.

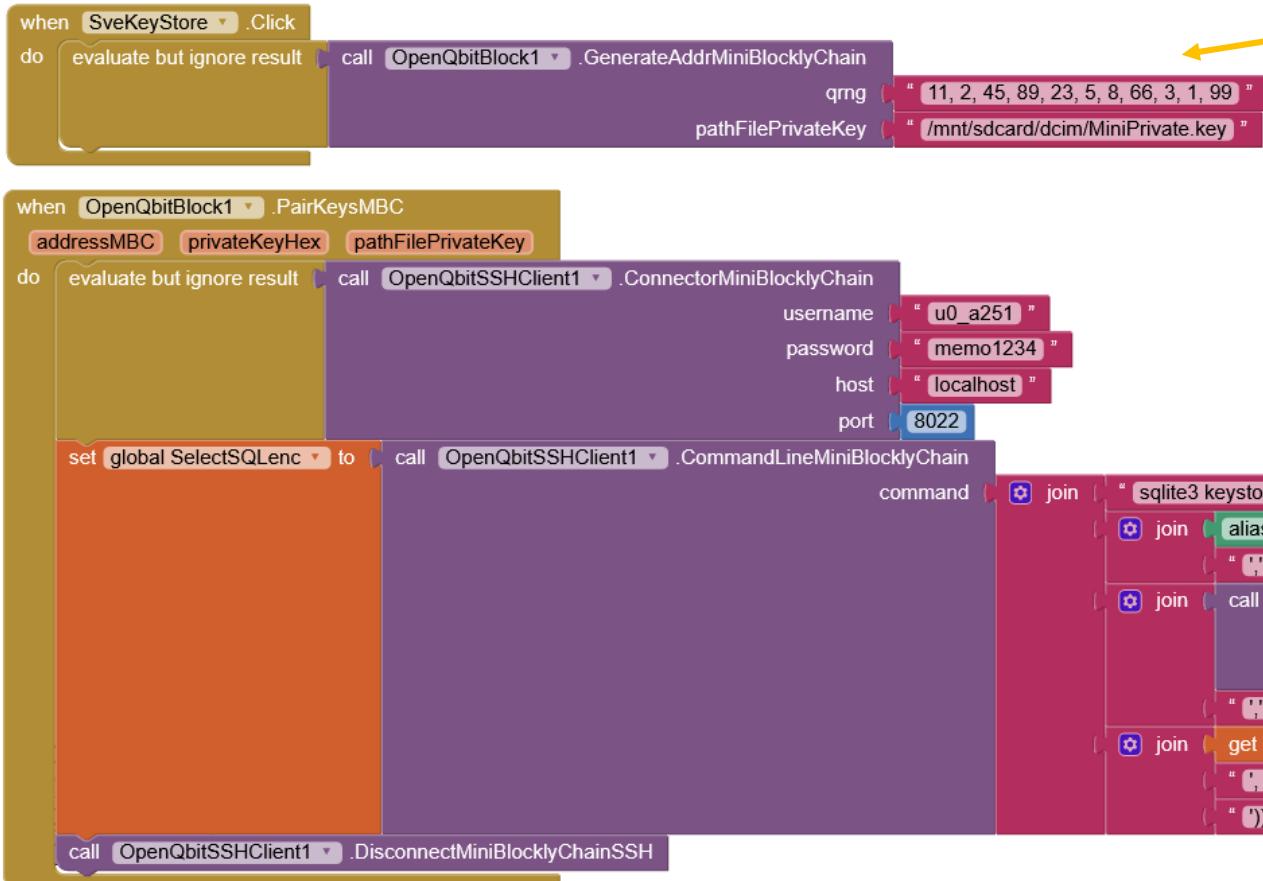
```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> CREATE TABLE privatekey (
...> id integer primary key AUTOINCREMENT,
...> alias VARCHAR(50) NOT NULL,
...> addrHex VARCHAR(65) NOT NULL,
...> addrMBC VARCHAR(65) NOT NULL,
...> addrBin BLOB NOT NULL
...> );
sqlite> .tables
privatekey
sqlite> .quit
$
```

Instruções SQL
 para criar uma
 nova tabela

Vamos agora utilizar algumas extensões de segurança para inserir dados de "chave privada" e depois consultar estes dados.

Utilizaremos o bloco para gerar um novo endereço de utilizador ([GenerateAddrMiniBlocklyChain](#)), este bloco dar-nos-á o endereço privado em dois formatos (hexadecimal e binário), bem como o endereço público em formato de endereço genérico MBC. Também utilizaremos a extensão ([OpenQbitSSHClient](#)) e a extensão ([OpenQbitEncDecData](#)) para ver mais detalhes destes, consulte a secção "Definição e utilização de blocos de segurança". No terminal Termux deverá estar a executar o serviço SSH.

INSERIR dados encriptados na base de dados **keystore.db**

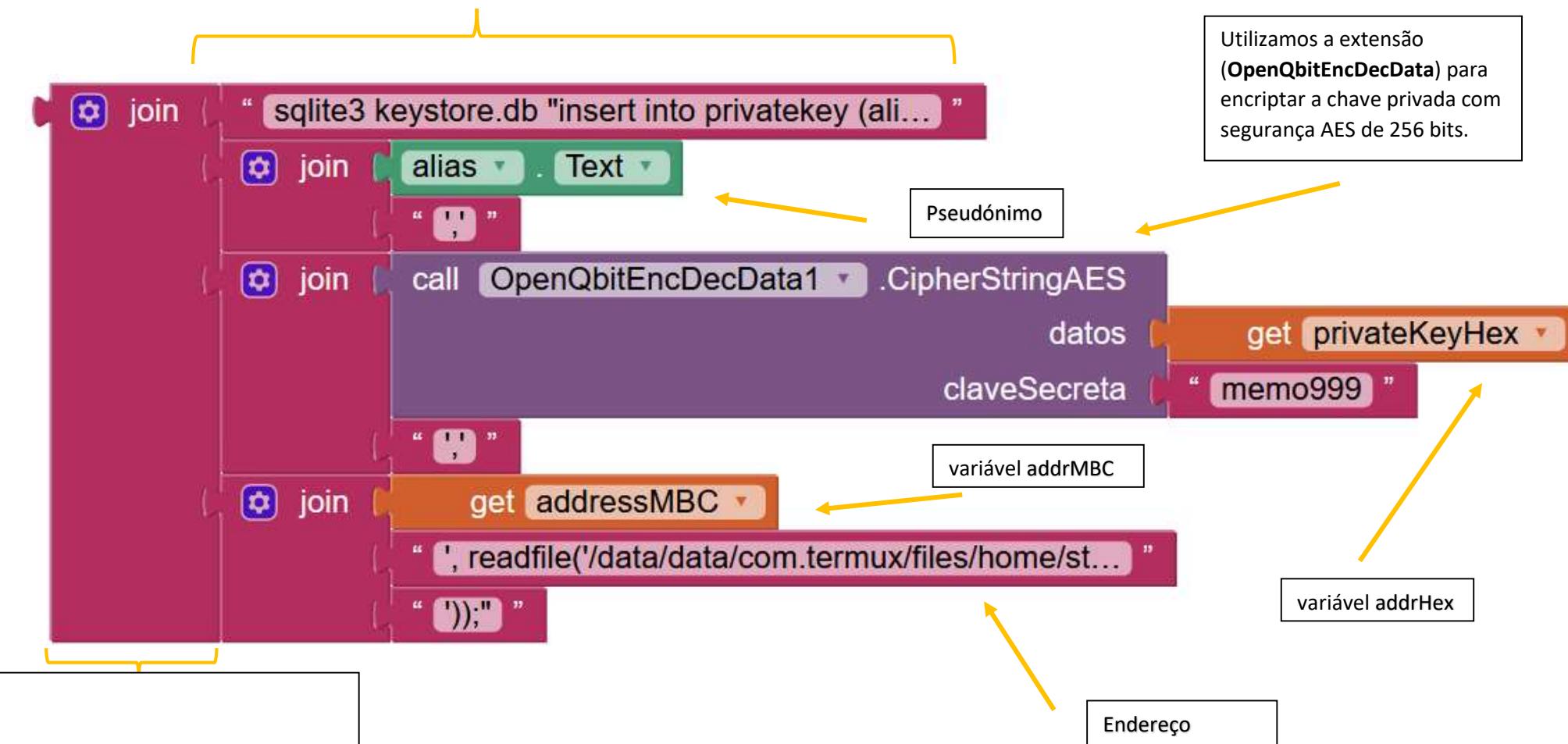


Séries de números quânticos aleatórios gerados pelo bloco (ApiGetQRNGInteger) verificar como formatar o resultado JSON, uma vez que a entrada do bloco (GenerateAddrMiniBlocklyChain) necessita de uma série de números apenas separados por

A sintaxe do comando é muito importante, temos de introduzir o seguinte comando no formato correcto no ambiente Blockly.

Os valores dos valores serão sempre as variáveis, exemplo:

```
sqlite3 keystore.db "inserir na chave privada (alias, addrHex, addrMBC, addrBin) valores ('memo', 'QWERTY', 'MBC12345',  
readfile('/data/data/com.termux/files/home/storage/dcim/MiniPrivate.key'))"
```



Após a execução dos blocos teremos um resultado na tabela de chave privada da base de dados da keystore.db semelhante ao seguinte:

Entramos com o manipulador de sqlite no terminal Termux, abrimos a base da keystore.db e executamos a frase:

```
$ sqlite3
```

SQLite versão 3.32.2 2020-06-20 15:25:24

Introduzir ".help" para dicas de utilização.

Ligado a uma base de dados in-memory transitória.

Utilizar ".open FILENAME" para reabrir numa base de dados persistente.

```
sqlite> .open keystore.db
```

```
sqlite> seleccionar * da chave privada;
```

```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> select * from privatekey;
1|mexico|O7JBBizTcC0Ce6aB8ve5aTV410l1DKiUQZyPSfJuRbVUZnvIwQJcry4LiiBM5jHavQeMo1iN8rC087V5Xka2YaKR4fswopeTQmI/Q+ipzI=|2eENpFt2HSuGtXNxoeiwfrp5d6e87Z7y5|000
sqlite>
```

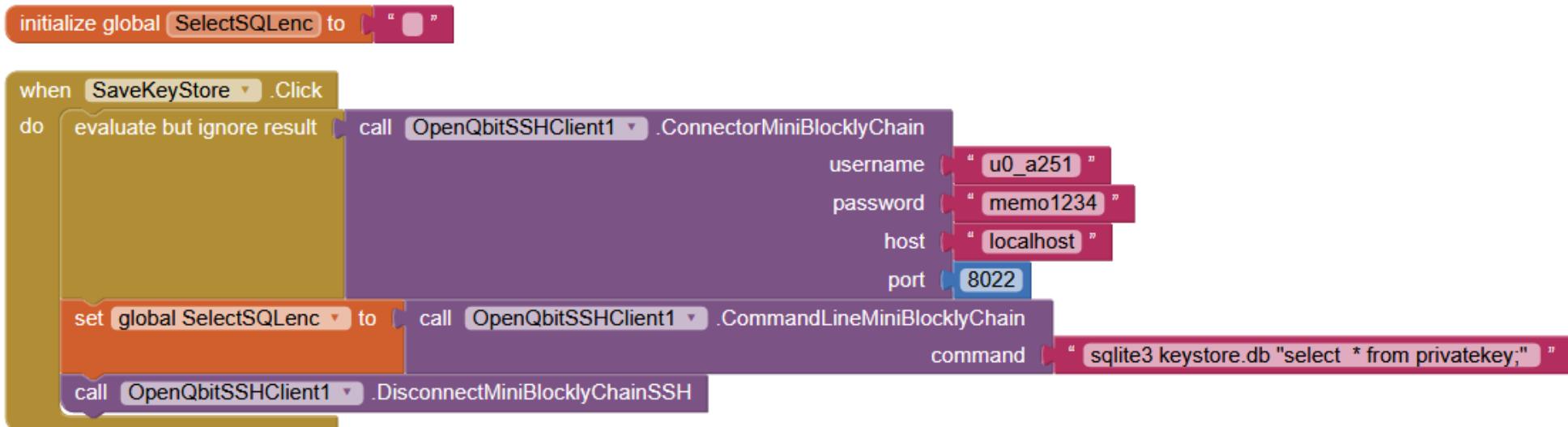
Pseudónimo:

addrBin (chave privada em

encriptação

endereçoMBC

CONSULTE todos os dados da tabela de **chave privada** da base de dados SQLite **keystore.db**



CONSULTE os pseudónimos dos dados na tabela de **chave privada** da base de dados SQLite **keystore.db**

sqlite3 keystore.db "select alias from privatekey;"

Consulta de todos os dados na coluna addrHex encriptada na tabela da **chave privada** da base de dados SQLite **keystore.db**

sqlite3 keystore.db "select addrHex from privatekey;"

CONSULTE para recuperar a chave privada addrBin na tabela de **chave privada** da base de dados SQLite **keystore.db**

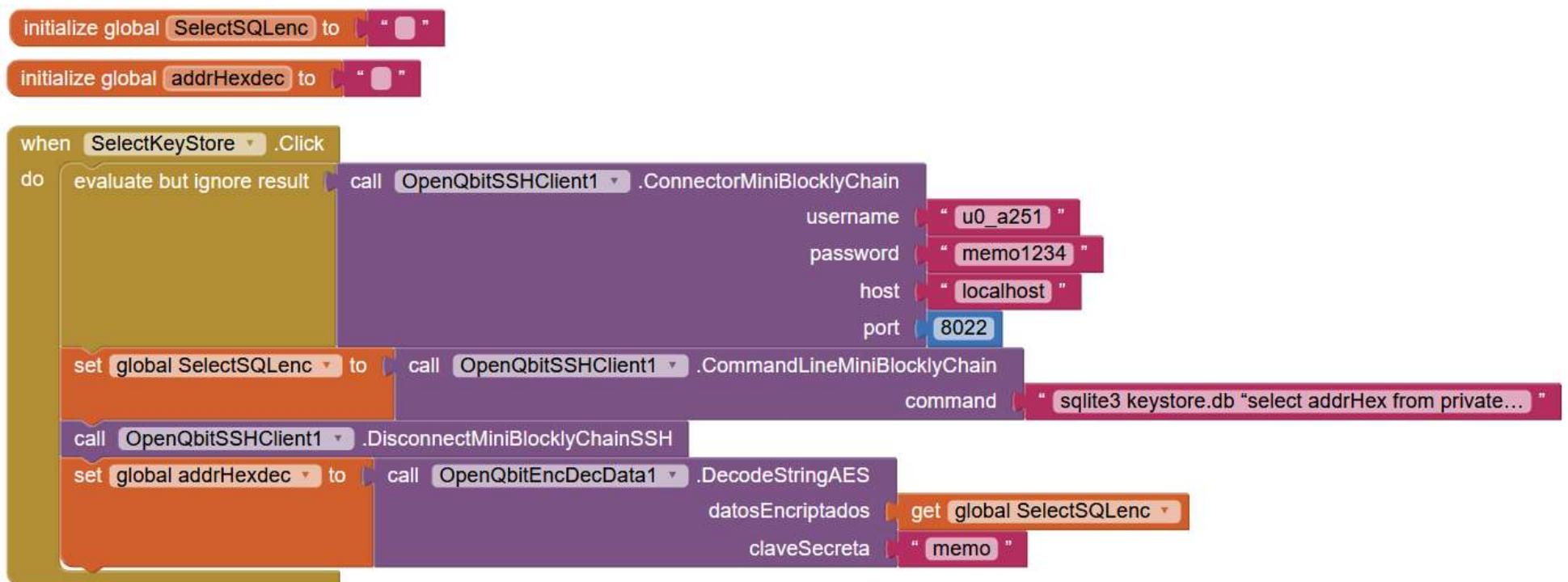
sqlite3 **writefile('PrivateKey.key', addrBin)** FROM privatekey WHERE alias='mexico'; (2)

(1) Este tipo de consulta será a principal para consultar a chave privada para realizar o processo de assinatura das transacções.

Consulta para Decodificação de Dados por Alias na coluna addrHex da tabela de **chave privada** da base de dados SQLite **keystore.db**

Neste caso, utilizaremos o bloco (**DecodeStringAES**) para decodificar os dados armazenados na coluna "addrHex".

sqlite3 keystore.db "select addrHex from privatekey where='mexico';"



Esta consulta dá-nos a chave privada em formato hexadecimal, esta é a parte fundamental de qualquer recepção ou envio de transacções. É repetidamente recomendado que seja mantida uma cópia de segurança desta base de dados keystore.db.

Concepção de bases de dados **publickeys.db** com tabela **publicaddr**:

\$ sqlite3

SQLite versão 3.32.2 2020-06-20 15:25:24

Introduzir ".help" para dicas de utilização.

Ligado a uma base de dados in-memory transitória.

Utilizar ".open FILENAME" para reabrir numa base de dados persistente.

sqlite> .open publickeys.db

sqlite> CREATE TABLE publicaddr (id integer primary key AUTOINCREMENT NOT NULL, pubMBC VARCHAR NOT NULL, pubHex VARCHAR NOT NULL, pubBin BLOB NOT NULL);

sqlite> .quit

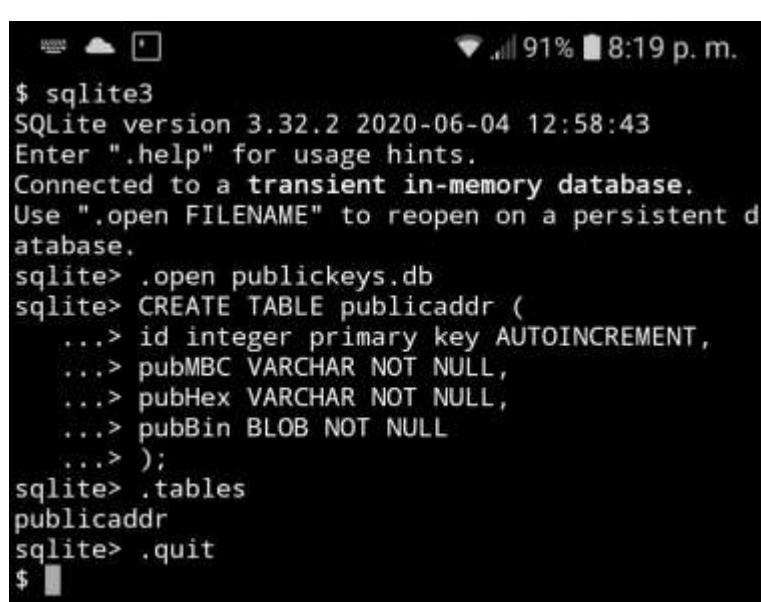
Em seguida, a criação da tabela **publishedaddr** é mostrada com a seguinte frase SQL em forma segmentada:

sqlite> CREATE TABLE publishedaddr (
...> id integer chave primária AUTOINCREMENTO
...> pubMBC VARCHAR NOT NULL,
...> pubHex VARCHAR NOT NULL,
...> pubBin BLOB BLOB NOT NULL
...>);

sqlite> .tabelas

publishedaddr

sqlite> .quit



The screenshot shows a terminal window on a mobile device. The status bar at the top indicates signal strength, battery level (91%), and the time (8:19 p.m.). The terminal prompt is '\$'. The user enters the command '\$ sqlite3' followed by the SQLite version information. Then, the user creates a table named 'publicaddr' with four columns: 'id' (integer primary key AUTOINCREMENT), 'pubMBC' (VARCHAR NOT NULL), 'pubHex' (VARCHAR NOT NULL), and 'pubBin' (BLOB NOT NULL). Finally, the user lists the tables in the database with '.tables' and exits the SQLite shell with '.quit'.

```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open publickeys.db
sqlite> CREATE TABLE publicaddr (
...> id integer primary key AUTOINCREMENT,
...> pubMBC VARCHAR NOT NULL,
...> pubHex VARCHAR NOT NULL,
...> pubBin BLOB NOT NULL
...> );
sqlite> .tables
publicaddr
sqlite> .quit
$
```

24. Anexo "Comandos RESTful SQLite GET/POST".

A seguir, são mostrados os diferentes comandos que podemos utilizar no SQLite Restful da rede de backup. Estes foram consultados a partir do desenvolvimento original do GITHUB (<https://github.com/olsonpm/sqlite-to-rest>)

Operações CRUD (Criar, Ler, Actualizar e Apagar) RESTful.

Segue-se uma lista de operações disponíveis disponibilizadas pela API RESTful sob a forma de pseudo exemplos. Assumiremos uma base "op.sqlite" e duas tabelas "trans" associadas para transacções e outro "sinal" para endereço de origem, endereço de destino e valor do activo com duas colunas id INTEGER PRIMARY KEY.

Como referido nas limitações, é favor notar que os métodos (DELETE e POST) só podem afectar uma fila de cada vez.

OBTENÇÃO Isto permite a maior variação. Mais tarde veremos todos os operadores de consulta disponíveis.

Os cabeçalhos podem ser especificados logo abaixo dos URLs.

/transRelementos

para todas as filas

/trans?id=1Onde

id = 1

/trans

intervalo: filas=0-2Primeira

três filas

/trans

intervalo: filas=-5Últimas

cinco filas

/trans

intervalo: filas=0-

Tantas filas quantas o servidor possa fornecer, que na prática será a menor das filas de maxRange e total de filas.

/trans

intervalo: filas=1-

Tantas filas quantas o servidor possa fornecer, a partir da fila 1.

/trans
ordem: nomeOrdenado
por nome ascendente

/trans
ordem: nome descendente ordenado
por nome descendente

/trans
ordem: nome desc, id

Contribuído, mas primeiro ordenado pelo nome descendente, e no caso de um empate por id ascendente.

/trans?id>1
Onde id > 1

/trans?id>==2&id<5
Onde id >= 2 e id < 5

/trans?name_NOTNULL
Onde o nome não é nulo

/trans?name_ISNULL Onde
o nome é nulo

/trans?id!=5&nome_LIKE'Spotted%'.
Onde id != 5 e nome é LIKE "Spotted%" (ignorar aspas)

/trans?id>==1&id<10&nome_LIKE'Avery%'.
ordem: nome desc, id intervalo: filas=2-4

Contribuído para dar o exemplo.

Obter transacção com identificadores entre 1 e 9 inclusive, com um nome como "Avery%", ordenado primeiro pelo nome decrescente e depois pelo identificador ascendente, obtendo a terceira a quinta fila do resultado. Ou em SQL:

DELETE Requer uma cadeia de consulta com todas as chaves primárias definidas igual a um valor. Isto requer uma eliminação máxima de uma linha.

/trans?id=1Deletes
trans com id=1

Se a transacção tivesse antes uma chave principal composta por identificação e nome.

/trans?id=1&name='Avery IPA'.

Criar POST

Não deve passar uma cadeia de perguntas. Se uma cadeia de consulta for passada, assume-se uma actualização POST. Todos os pedidos POST devem passar o tipo de conteúdo do cabeçalho: pedido / json.

Note-se que o corpo deve conter todas as colunas PRIMARY KEY não canceláveis e não INTEGRAR. Caso contrário, será enviada uma resposta de 400 indicando quais os campos que foram perdidos. As colunas nulas serão nulas e as colunas INTEGER PRIMARY KEY serão automaticamente aumentadas de acordo com as especificações sqlite3.

Os dados do JSON serão especificados logo abaixo dos URLs.

/trans

```
{"id":1,"name":"Serendipity"}Cria  
um trans com id = 1 e nome = "Serendipity".
```

/trans

```
{"id":1}Cria  
um trans com id = 1 e nome = NULL
```

/trans

```
{"nome": "Serendipidade"}
```

Cria uma transacção com conjunto de id para o seguinte valor aumentado por sqlite3 Especificações CHAVE PRIMÁRIA INTEGRAL.

/trans

```
{}
```

Cria uma transacção com identificação aumentada e o nome definido para NULL.

Actualização do POST

Deve conter uma cadeia de consulta. Sem uma cadeia de consulta, assume-se a criação do POST. Tal como no POST create, o tipo de conteúdo do cabeçalho: aplicação / json é necessário.

A cadeia de consulta deve conter todas as chaves primárias para assegurar que apenas uma linha seja actualizada. Se forem passados valores incorrectos, será devolvido um 400 com as chaves ofensivas.

O corpo do pedido deve conter um objecto não vazio e deve conter chaves válidas correspondentes aos nomes das colunas.

Os dados do JSON serão especificados logo abaixo dos URLs.

/trans?id=1

```
{"id":2}
```

OpenQbit.com

Actualizar a transacção com um ID de 1, definindo-a para dois.

/trans?id=1
{ "nome" : " MCBza45Rt56cvbgfdR2Swd788kj" }

Actualizar a transacção com o ID de 1 definindo o seu nome ou valor para "MCBza45Rt56cvbgfdR2Swd788kj".

Se em vez disso a mesa de comércio tivesse uma chave principal composta por identificação e nome.

/trans?id=1&name=MCBza45Rt56cvbgfdR2Swd788kj
{ "nome" : " MCB3ofFG5Hj678MNb09vLdfaasx " }

Actualizar a transacção onde id é um e o endereço é MCBza45Rt56cvbgfdR2Swd788kj, configurando o MCB3ofFG5Hj678MNb09vLdfaasx.

Referência

isSqliteFile

Basta verificar os primeiros 16 bytes do ficheiro para ver se é igual a 'formato sqlite 3' seguido de um byte nulo.

isDirectory

Retorna o resultado do fs.statsSync seguido de .isDirectory

isFile

Retorna o resultado do fs.statsSync seguido de .isFile

Consultar operadores de GET

As condições de consulta devem ser marcadas com símbolos de ligação, por exemplo id> 5 & nome = MCBza45Rt56cvbgfdR2Swd788kj

Operadores binários (requer um valor depois) Homem.

=
!=
>=
<=
>
<
LIKE

LIKE é especial porque deve ter citações simples de abertura e fecho. Caso contrário, será gerado um erro de 400 mostrando onde a análise não pôde ser concluída e o que era esperado. Ver CRUD RESTful Operations para exemplos.

Operadores individuais (devem seguir o nome de uma coluna)

É NULO

NÃO NULL

Objecto de configuração do router

isLadenPlainObject

O objectivo deste objecto é fornecer uma configuração genérica para o router sqlite. São permitidas as seguintes propriedades:

prefixo: isLadenString A corda passou para a opção construtor de prefixos koa-router. Por exemplo, o servidor esqueleto não especifica um prefixo, o que permite que a API da cerveja seja atingida directamente da raiz do domínio http://localhost: 8085 / trans. Se definir o prefixo '/ api', então deve enviar pedidos para http://localhost: 8085 / api / trans.

allTablesAndViews: um objecto de configuração tabular

As configurações especificadas neste objecto serão aplicadas a todas as tabelas e vistas, opcionalmente substituídas pela propriedade tabelasAndViews.

tablesAndViews: isLadenPlainObject O objecto passado deve ter chaves que coincidam com a coluna da base de dados ou ver os nomes. Caso contrário, será emitida uma mensagem de erro amigável. Os valores para cada tabela e vista devem ser um objecto de configuração tabular.

Objecto de configuração tabular

isLadenPlainObject Este objecto representa configurações que podem ser definidas para vistas ou tabelas. Permite as seguintes propriedades:

maxRange: isPositiveNumber

Aplicação por defeito: 1000

Este é o intervalo máximo que o seu servidor permitirá pedidos. Se um pedido de GET chega sem um cabeçalho de alcance, a especificação assume que deseja o recurso completo. Se o número de linhas que resultam em GET for superior ao maxRange, é devolvido um status 416 com o cabeçalho personalizado de max-range. O valor por defeito da aplicação é deliberadamente conservador na esperança de que os autores definam o maxRange de acordo com as suas necessidades.

Note-se que 'Infinity' é um número positivo válido.

bandeiras: isLadenArray

Actualmente, o único indicador aceite é a string 'sendContentRangeInHEAD'. Quando definido, os pedidos HEAD devolverão a gama de conteúdos disponíveis na forma - gama de conteúdos: * / <max gama>. A razão pela qual é configurável é que o cálculo do intervalo máximo pode ser mais trabalhoso do que vale, dependendo da carga do servidor e do tamanho das suas tabelas.

Cabeçalhos personalizados

Aplicação

ordem: este cabeçalho só é definido para GET, e pode ser considerado como o equivalente a sql ORDEM POR. Deve conter um nome de coluna delimitado por vírgulas, cada uma opcionalmente seguida de um espaço e as cordas 'asc' ou 'desc'. Se forem enviados valores de encomenda incorrectos, uma resposta de 400 indicará quais.

Resposta

Nem todos são necessariamente personalizados, mas toda a utilização está fora da especificação e, por conseguinte, necessita de ser esclarecida.

GET

max-range: este cabeçalho é devolvido quando o número de filas solicitadas excede o maxRange configurado. Note-se que o pedido pode não especificar o cabeçalho do intervalo, mas o número de filas que resultam nesse recurso ainda será verificado.

gama de conteúdos: rfc7233 estados

Apenas os códigos de estado 206 (Conteúdo Parcial) e 416 (Intervalo Insatisfatório) descrevem um significado para Content-Range.

Quando o sqlite-to-rest responde com um código de estado 200, o cabeçalho da gama de conteúdos é enviado com o formato 206 de <linha início> - <linha fim> / <linha contagem>.

Quando um pedido é enviado sem um cabeçalho de intervalo e o número de linhas resultante excede o intervalo máximo, um 400 é devolvido com o intervalo de conteúdo definido para o formato 416 de * / <contagem de linhas>

Note-se que este cabeçalho pode ser devolvido numa resposta HEAD.

ordem de aceitação: será devolvida se a ordem do cabeçalho do pedido tiver uma sintaxe incorrecta ou nomes de colunas incorrectos especificados. Para mais detalhes, ver HEAD -> accept-order abaixo.

25. Anexo "Conector SQLite-Redis de código Java".

Um código da ligação entre a comunicação de ambas as bases de dados SQLite-Redis é mostrado abaixo. Basicamente, como podemos ver, o conector muda o formato SQLite para uma sequência genérica dos dados (transacções) para que a Redis receba.

O processo acima referido actua como um gatilho de fila de transacção para todos os nós que estão ligados e são possíveis candidatos para processar a fila de transacção actual.

Quando a base de dados Redis receber a fila de transacções pela configuração Mestre-escravo que possui, distribuirá a informação em tempo real e igualmente a todos os nós.

Outro ponto fundamental é que todos os nós devem ser sincronizados no seu relógio, isto será feito como vimos anteriormente com a funcionalidade da consulta a um servidor NTP (Network Time Protocol).

Este conector também executa o primeiro nível de revisão de segurança de dados, calculando a árvore Merkle Root de todas as transacções.

Código base do SQLite-Redis Connector.

```
importação java.sql.*;
importação java.util.*;
importação de java.util.arrays;
importação java.util.list;
importação java.util.array;
importação java.security.*;
importação java.security.MessageDigest;
import redis.clients.jedis.Jedis;
classe RediSqlite{
    public String previousHash;
    pública String merkleRoot;
    public static ArrayList<String> transaction = novo ArrayList<String>();
    public static ArrayList<String> treeLayer = nova ArrayList<String>();
    public static String getMerkleRoot() {
        int count = transaction.size();
        int item = 1;
        Int ímpar = 0;
        ArrayList<String> previousTreeLayer = transacções;
        while(count > 1) {
            treeLayer = novo ArrayList<String>();
```

```

for(int i=1; i <= count ; i=i+2) {

    if (!esPar(count) && i ==contar) odd = 1;
        treeLayer.add(applySha256(previousTreeLayer.get(i-item)
previousTreeLayer.get(i-impar))) + 

    }

item = 1;
ímpar = 0;
count = treeLayer.size();
previousTreeLayer = treeLayer;
}

String merkleRoot = (treeLayer.size() == 1) ? treeLayer.get(0) : " ";
    devolver merkleRoot;
}

//Definir se a Merkle Tree é par ou ímpar
boolean estático enPar(número int){
    se (numero%2==0) voltar verdadeiro; caso contrário, voltar falso;
}

public static String applySha256(String input){
    tente {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        //Aplica sha256 à nossa contribuição,
        byte[] hash = digest.digest(input.getBytes("UTF-8"));
        StringBuffer hexString = novo StringBuffer(); // Isto conterá hash como hexadecimal
        para (int i = 0; i < hash.length; i++) {
            String hex = Integer.toHexString(0xff & hash[i]);
            if(hex.length() == 1) hexString.append('0');
            hexString.append(hex);
        }
        devolver hexString.toString();
    }
    catch(Excepção e) {
        lançar nova RuntimeException(e);
    }
}

vazio estático público principal(String args[]){
    tentar{

```

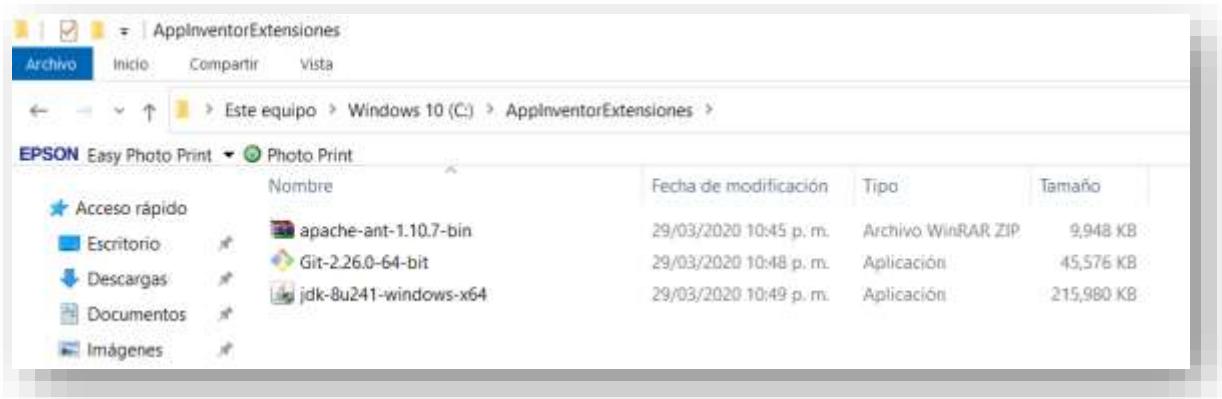
```
Connection con=DriverManager.getConnection("jdbc:sqlite:C:/memo/sqlite-tools-
win32-x86-3310100/trans.sqlite3");
//Conexão ao servidor Redis no localhost
Jedis = novo Jedis ("localhost");
jedis.auth("memo1234");
Declaração stmt=con.createStatement();
String sql = "SELECT * FROM cervejaria";
ResultSet rs=stmt.executeQuery(sql);
while(rs.next()) {
    transactions.add(rs.getString(2));
    jedis.set("LATAM:"+String.valueOf(rs.getInt(1)), "["+"\""+rs.getString(2)+"\""+","+"\""+rs.get
String(3)+"\""+","+"\""+rs.getString(4)+"\"");
}
jedis.set("LATAM:merkleroot", getMerkleRoot());
System.out.println("Número de elementos da ArrayList:: "+treeLayer.size()));
com .close();
}catch(Excepção e){ System.out.println(e);}
}

}
```

26. Anexo "Mini BlocklyChain para programadores".

Neste anexo iremos rever a configuração, instalação e utilização básica de como gerar módulos externos baseados na linguagem de programação Java para ambiente Blockly e poderemos criar módulos especializados para cada caso de negócio e inserir funcionalidades no Sistema Mini BlocklyChain ou adicionar outras funcionalidades à nossa aplicação móvel.

Criámos um directório no nosso sistema Windows chamado "ApplInventorExtensions" e este irá descarregar os seguintes pacotes de software públicos.



1.- Vamos descarregar a última versão do **JDK (Java Development Kit)**

exemplo: jdk-8u251-windows-x64.exe (211 MB)

<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>



2.- Biblioteca Apache Ant que usa JAVA para construir aplicações, <http://ant.apache.org/bindownload.cgi>, no meu caso descarreguei Ant 1.10.8 (Distribuições Binárias) (apache-ant-1.10.8-bin.zip). Pode haver versões mais avançadas.

1.9.15 release - requires minimum of Java 5 at runtime

- 1.9.15 .zip archive: [apache-ant-1.9.15-bin.zip \[PGP\] \[SHA512\]](#)
- 1.9.15 .tar.gz archive: [apache-ant-1.9.15-bin.tar.gz \[PGP\] \[SHA512\]](#)
- 1.9.15 .tar.bz2 archive: [apache-ant-1.9.15-bin.tar.bz2 \[PGP\] \[SHA512\]](#)

1.10.8 release - requires minimum of Java 8 at runtime

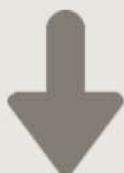
- 1.10.8 .zip archive: [apache-ant-1.10.8-bin.zip \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.gz archive: [apache-ant-1.10.8-bin.tar.gz \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.bz2 archive: [apache-ant-1.10.8-bin.tar.bz2 \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.xz archive: [apache-ant-1.10.8-bin.tar.xz \[PGP\] \[SHA512\]](#)

Old Ant Releases

Older releases of Ant can be found [here](#). We highly recommend to not use those releases but upgrade to Ant's [latest](#) release.

3.- Instalámos o Git Bash a partir do seu site <https://git-scm.com/download/win>

Downloading Git



You are downloading the latest (**2.27.0**) **64-bit** version of **Git for Windows**. This is the most recent [maintained build](#). It was released **9 days ago**, on 2020-06-01.

[Click here to download manually](#)

Other Git for Windows downloads

[Git for Windows Setup](#)

[32-bit Git for Windows Setup](#).

[64-bit Git for Windows Setup](#).

[Git for Windows Portable \("thumbdrive edition"\)](#)

[32-bit Git for Windows Portable](#).

[64-bit Git for Windows Portable](#).

The current source code release is version **2.27.0**. If you want the newer version, you can build it from [the source code](#).

4.- Descompactar "Apache Ant." Quando terminar de descompactar, pode fazê-lo numa pasta dupla, por exemplo:

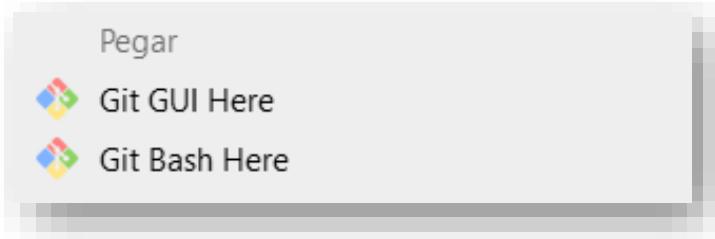
C:\AppInventorExtensions\apache-ant-1.10.8-bin\apache-ant-1.10.8-bin

- Alterar para C:\Apache-ant-1.10.8-bin

Nombre	Fecha de modificación	Tipo	Tamaño
bin	01/09/2019 11:43 a. m.	Carpeta de archivos	
etc	01/09/2019 11:43 a. m.	Carpeta de archivos	
lib	01/09/2019 11:43 a. m.	Carpeta de archivos	
manual	01/09/2019 11:43 a. m.	Carpeta de archivos	
CONTRIBUTORS	01/09/2019 11:43 a. m.	Archivo	7 KB
contributors	01/09/2019 11:43 a. m.	Documento XML	33 KB
fetch	01/09/2019 11:43 a. m.	Documento XML	14 KB
get-m2	01/09/2019 11:43 a. m.	Documento XML	5 KB
INSTALL	01/09/2019 11:43 a. m.	Archivo	1 KB
KEYS	01/09/2019 11:43 a. m.	Archivo	94 KB
LICENSE	01/09/2019 11:43 a. m.	Archivo	15 KB
NOTICE	01/09/2019 11:43 a. m.	Archivo	1 KB
patch	01/09/2019 11:43 a. m.	Documento XML	2 KB
README	01/09/2019 11:43 a. m.	Archivo	5 KB
WHATSNEW	01/09/2019 11:43 a. m.	Archivo	250 KB

5.- Instalámos o Git Bash. Deixámos tudo por defeito na instalação.

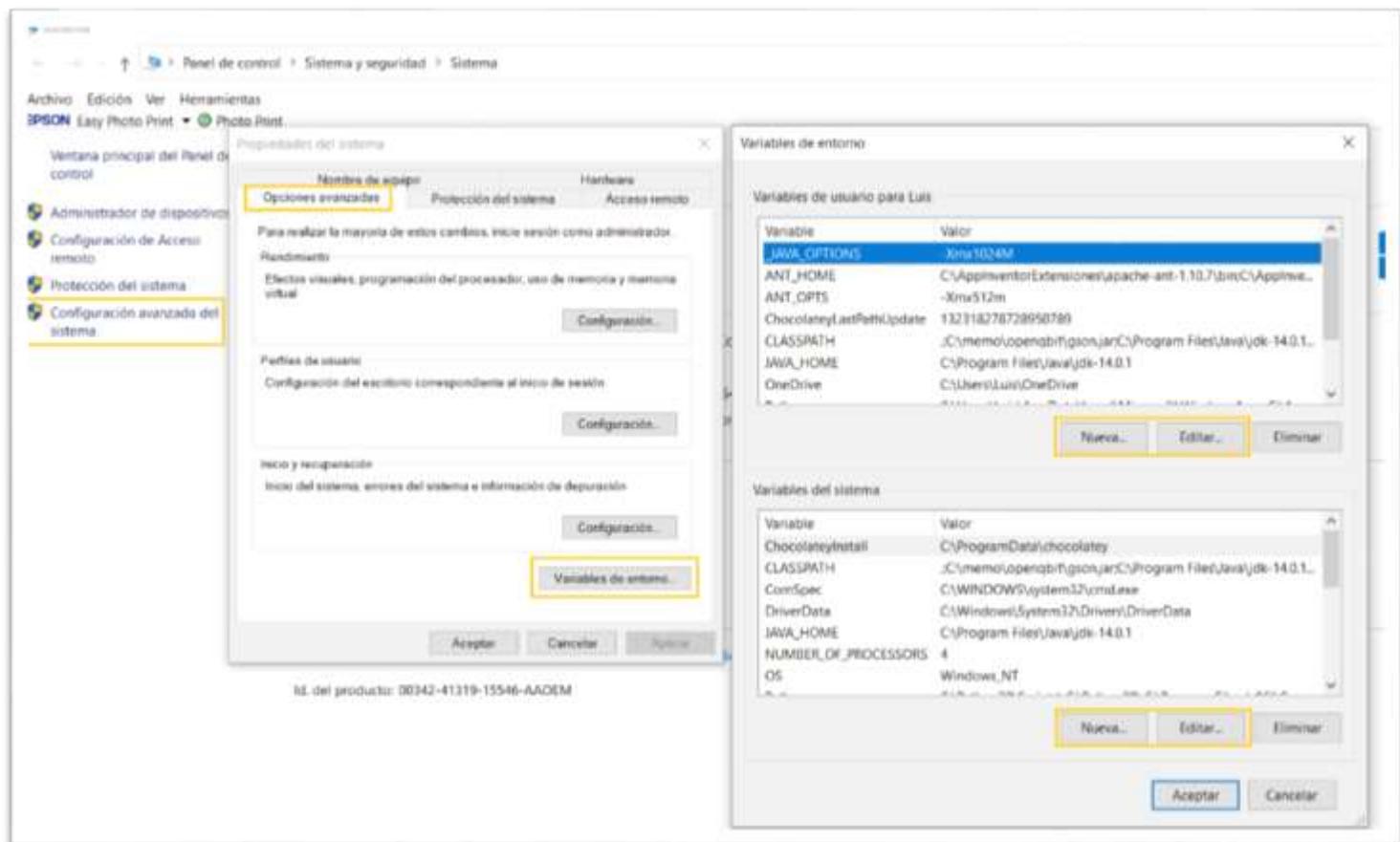
- Mais tarde podemos ver que tem um link no Botão Iniciar do Windows, clicando no botão da esquerda no navegador de ficheiros.



6.- Instalámos o Java SE Development Kit. Dependendo da versão do Windows, provavelmente precisará de reiniciar o seu computador.

Variáveis de ambiente de sistema Windows. Iremos criar as variáveis ambientais. Para o fazermos, iremos:

Painel de Controlo -> Sistema -> Configurações Avançadas do Sistema -> Opções Avançadas -> Variáveis de Ambiente.



Dependendo se queremos colocar uma nova variável ou editar uma já existente, carregamos no botão "New..." ou "Edit...".

Para acrescentar endereços aos já estabelecidos, separamo-los por ponto-e-vírgula;

Na secção de Variáveis de Utilizador para John, criamos estas novas...

JAVA_OPTIONS colocamos-lhe de Valor -Xmx1024m

ANT_HOME pomos de valor C:\AplicInventorExtensions\apache-ant-1.10.8-bin [ou seja, a pasta onde descomprimimos o apache-ant]

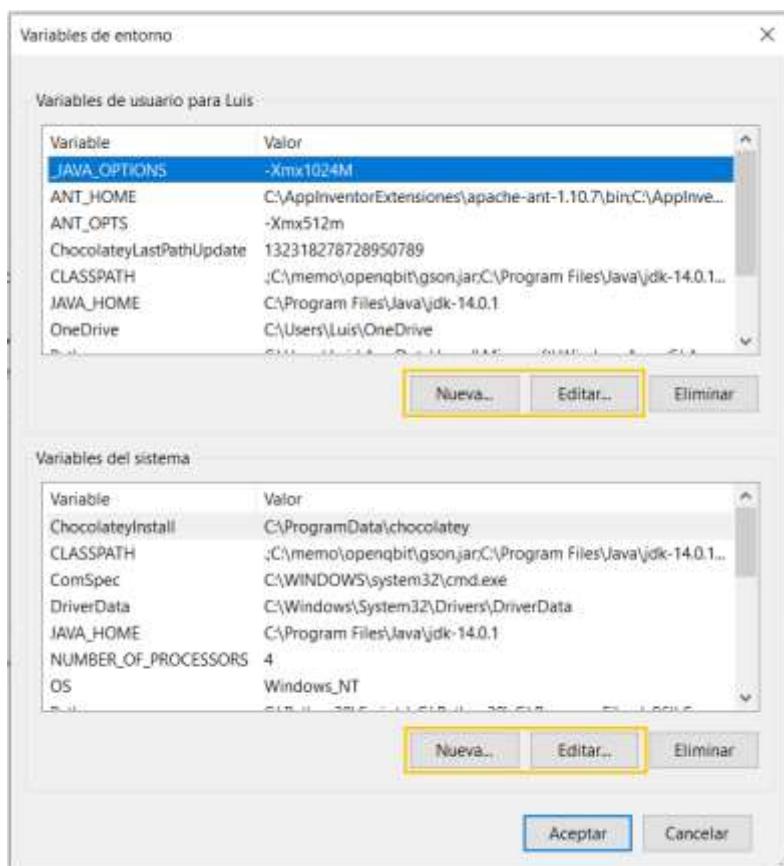
ANT_OPTS que colocamos de valor -Xmx256M

JAVA_HOME está definido para C:\Program Files\jdk1.8.0_131 [Se tiver outro valor, altere-o. Note que é jdk NOT jdr]

CLASSPATH colocamos de Valor %ANT_HOME%\lib;%JAVA_HOME%\lib

Em PATH adicionámos ;%ANT_HOME%\bin;%JAVA_HOME%\bin [Note-se que ;começa com um ponto-e-vírgula; para adicionar aos que já lá estão].

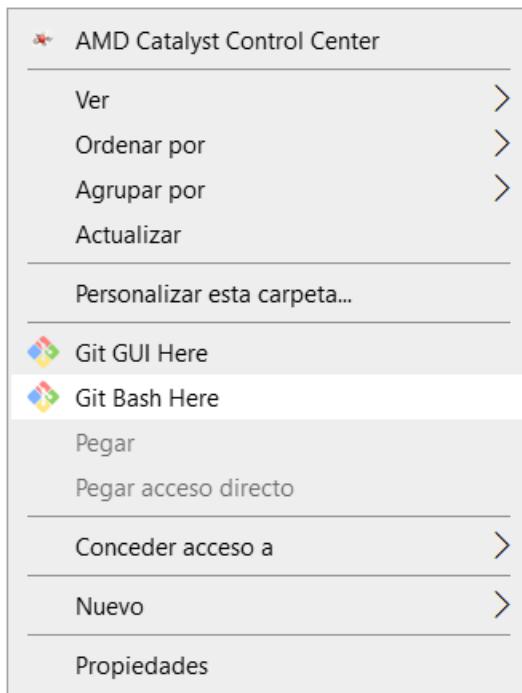
NOTA: As variáveis estão separadas umas das outras por ponto e vírgula: variable-one; variable-n; variable-n+1



7.- Criação do clone do App Inventor no nosso computador

Vamos criar uma cópia "clone" de App Inventor no nosso servidor (PC), descarregá-la directamente da Internet e criar essa cópia.

Para o fazer, utilizaremos a aplicação **Git Bash** e clicaremos sobre ela para abrir um terminal.



Executamos o comando no terminal de Git Bash:

Clone de idiota <https://github.com/mit-cml/appinventor-sources.git>

```
Administrator: ~
```

```
git clone https://github.com/mit-cml/appinventor-sources.git
Cloning into 'appinventor-sources'...
remote: Counting objects: 43192, done.
remote: Total 43192 (delta 0), reused 0 (delta 0), pack-reused 43190
receiving objects: 100% (43192/43192) 551.30 MiB | 404.00 KiB/s, done.
resolving deltas: 100% (27590/27590)  done.
Checking out files: 100% (27590/27590)  done.
```

```
Administrator: ~
```

O sitio onde se encontra o repositório:

<https://github.com/mit-cml/appinventor-sources/>

Irá criar uma pasta chamada appinventor-sources com a fonte App Inventor.

Nombre	Fecha de modificación	Tipo	Tamaño
.github	30/03/2020 01:05 a. m.	Carpeta de archivos	
appinventor	03/05/2020 05:32 p. m.	Carpeta de archivos	
.gitmodules	30/03/2020 01:05 a. m.	Documento de tex...	1 KB
bootstrap	30/03/2020 01:05 a. m.	Shell Script	2 KB
LICENSE	30/03/2020 01:05 a. m.	Archivo	12 KB
README.md	30/03/2020 01:05 a. m.	Archivo MD	11 KB
sample-	30/03/2020 01:05 a. m.	Documento de tex...	1 KB
Vagrantfile	30/03/2020 01:05 a. m.	Archivo	1 KB

Criámos o seguinte directório no caminho C: Utilizadores - Apppinventor-sourcesv-babkup - Apppinventor-components -rc-openqbit

No interior copiamos o nosso seguinte programa de teste chamado OpenQbitQRNGch.java

Nombre	Fecha de modificación	Tipo	Tamaño
OpenQbitQRNGch	11/06/2020 01:39 a. m.	Archivo JAVA	5 KB

Criação da extensão.

RESPEITANDO OS CAPS E OS MINUSCULOS, não é a mesma coisa Olá e Olá.

Não colocar qualquer acento. Estamos prontos para fazer a nossa primeira extensão. Será o gerador de números quânticos aleatórios.

Utilizamos um Editor de Texto, Notepad++, criamos um ficheiro chamado OpenQbitQRNGch.java que gera números quânticos aleatórios com o seguinte código:

```
// Esta extensão é utilizada para produzir o Quantum Random Number
Generator QRNG Switzerland API.

pacote com.openqbit. OpenQbitQRNGch;
importar com.google.appinventor.componentes.anotações.DesignerComponent;
importar com.google.appinventor.components.annotations.DesignerProperty;
importar com.google.appinventor.components.annotations.PropertyCategory;
importar com.google.appinventor.components.annotations.SimpleEvent;
importar com.google.appinventor.components.annotations.SimpleFunction;
importar com.google.appinventor.components.annotations.SimpleObject;
importar com.google.appinventor.components.annotations.SimpleProperty;
importar com.google.appinventor.components.common.ComponentCategory;
importar com.google.appinventor.components.common.PropertyTypeConstants;
importar com.google.appinventor.components.runtime.util.mediaUtil;
importar com.google.appinventor.components.runtime.*;
importação java.io.*;

@DesignerComponent(version = OpenQbitQRNGch.VERSION,
    descrição = "API Quantum Random Number Generator". Quantum random
numbers as a service, QRNGaaS, web API para o gerador de números
quânticos aleatórios do Quantis desenvolvido pela empresa suíça - " +
"Guillermo Vidal - OpenQbit.com",
    categoria = ComponenteCategoria.EXTENSÃO
    nãoVisível = verdadeiro,
    iconName = "http://www.pinntar.com/logoQbit.png")
@SimpleObject(externo = verdadeiro)

classe pública OpenQbitQRNGch estende AndroidNonvisibleComponent
implementa Componente {

    público estático final int VERSÃO = 1;
    público estático final String DEFAULT_TEXT_1 = "";
    contentor privado ComponenteContainer;
    texto privado String_1 = "";

    openQbitQRNGch(ComponentContainer container) público {
        super(recipiente.$form());
        this.container = contentor;
    }

    // Estabelecimento dos Imóveis.
    //@DesignerProperty(editorType =
    PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue =
    OpenQbitQRNGch.DEFAULT_TEXTO_1 + "")
    //@SimpleProperty(descrição = "API Get Quantum Random Number Generator,
    número aleatório entre 0 e 1. Introduzir variável *quantidade* de números
    a gerar")

    @SimpleFunction(descrição = "API Get Quantum Random Number Generator,
    número aleatório entre 0 e 1. Introduzir número inteiro variável
    *quantidade* de números a gerar")
    pública String APIGetQRNGdecimal(String qty) lança Excepção {
        String url = "http://random.openqu.org/api/rand?size=" + qty;
```

```

Comando String[] = {"curl", url };

ProcessBuilder process = novo ProcessBuilder(comando);
Processo p;
p = process.start();
BufferedReader reader = novo BufferedReader(novo
InputStreamReader(p.getInputStream()));
StringBuilder construtor = novo StringBuilder();
Linha de corda = nula;
enquanto ((line = reader.readLine()) != null) {
    builder.append(line);

    builder.append(System.getProperty("line.separator")));
}
String resultado = builder.toString();
//System.out.print(resultado);
resultado do retorno;

}

@SimpleFunction(descrição = "API Get Quantum Random Number Generator",
número aleatório entre um intervalo de min a max números. Introduzir
número inteiro variável *quantidade* de números para gerar um intervalo
de número mínimo *min* e máximo *número máximo *max*)
pública String APIGetQRNGinteger(String qty, String min, String max)
lança Excepção {
    String url = "http://random.openqu.org/api/randint?size=" + qty +
"&min=" + min + "&max=" + max
    Comando String[] = {"curl", url };

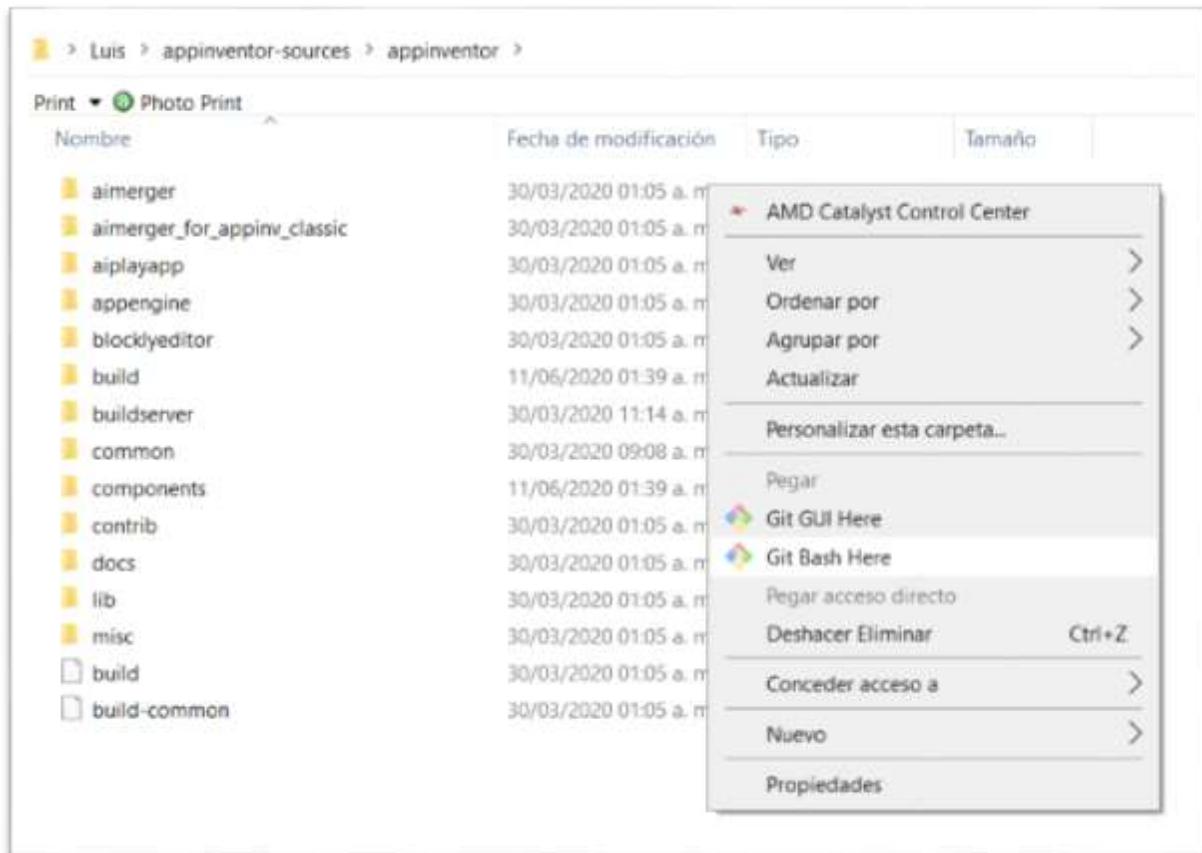
    ProcessBuilder process = novo ProcessBuilder(comando);
    Processo p;
    p = process.start();
    BufferedReader reader = novo BufferedReader(novo
InputStreamReader(p.getInputStream()));
    StringBuilder construtor = novo StringBuilder();
    Linha de corda = nula;
    enquanto ((line = reader.readLine()) != null) {
        builder.append(line);

        builder.append(System.getProperty("line.separator")));
    }
    String resultado = builder.toString();
    //System.out.print(resultado);
    resultado do retorno;

}
}

```

Executamos o **Git Bash** posicionando-nos no seguinte caminho: **C: Luis-appinventor-sources-appinventor**. Neste directório, clicamos no botão esquerdo do rato e seleccionamos o terminal **Git Bash**.



O terminal Git bash será posicionado no seguinte directório:

C:\Utentes\\iSuis\iSappinventor-fontes\iSappinventor (mestre)

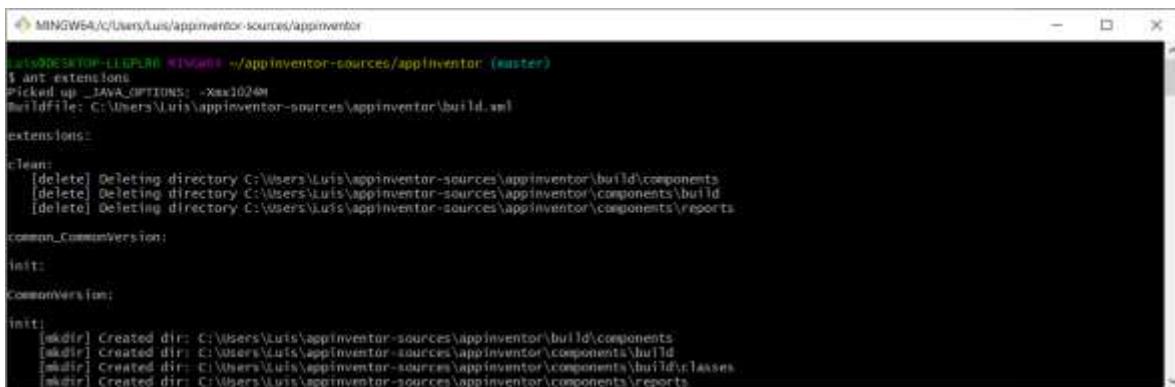
```

MINGW64/c/Users/Luis/appinventor-sources/appinventor
Luis@DESKTOP-LLGPLR6 MINGW64 ~ /appinventor-sources/appinventor (master)
$ |

```

No terminal de Git Bash e executar o seguinte comando:

Extensões de formigas \$



```
MINGW64/C:/Users/Luis/appinventor-sources/appinventor
luis@DESKTOP-LI6UPRI MINGW64 ~/appinventor-sources/appinventor (master)
$ ant extensions
Picked up _JAVA_OPTIONS: -Xmx1024M
Buildfile: C:/Users/Luis/appinventor-sources/appinventor/build.xml

extensions:

clean:
[delete] Deleting directory C:/Users/Luis/appinventor-sources/appinventor/build/components
[delete] Deleting directory C:/Users/Luis/appinventor-sources/appinventor/components/build
[delete] Deleting directory C:/Users/Luis/appinventor-sources/appinventor/components/reports

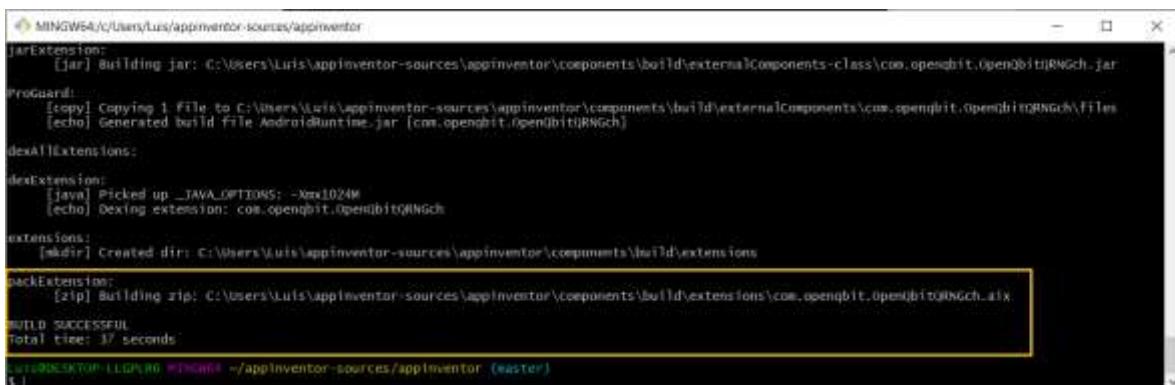
common_CommonVersion:
init:
CommonVersion:
init:
[mkdir] Created dir: C:/Users/Luis/appinventor-sources/appinventor/build/components
[mkdir] Created dir: C:/Users/Luis/appinventor-sources/appinventor/components/build
[mkdir] Created dir: C:/Users/Luis/appinventor-sources/appinventor/components/build/classes
[mkdir] Created dir: C:/Users/Luis/appinventor-sources/appinventor/components/reports
```

Se tudo corresse bem, consegui-lo-emos: CONSTRUIR COM SUCESSO.

A nossa extensão foi criada em...

C:\Utentes\Luis-appinventor-fontes-pinvendor-componentes-construção de extensões

NOTA: o conteúdo da pasta das extensões é apagado e recriado cada vez que fazemos uma extensão de formiga.



```
MINGW64/C:/Users/Luis/appinventor-sources/appinventor
jarExtension:
[jar] Building jar: C:/Users/Luis/appinventor-sources/appinventor/components/build/externalComponents-class.jar
ProGuard:
[copy] Copying 1 File to C:/Users/Luis/appinventor-sources/appinventor/components/build/externalComponents/com.openqbit.OpenQbitQRNGch/
[echo] Generated build file AndroidRuntime.jar (com.openqbit.OpenQbitQRNGch)
dexAllExtensions:
dexExtension:
[java] Picked up _JAVA_OPTIONS: -Xmx1024M
[echo] Dexing extension: com.openqbit.OpenQbitQRNGch
extensions:
[mkdir] Created dir: C:/Users/Luis/appinventor-sources/appinventor/components/build/extensions
packExtension:
[zip] Building zip: C:/Users/Luis/appinventor-sources/appinventor/components/build/extensions/com.openqbit.OpenQbitQRNGch.aix
BUILD SUCCESSFUL
Total time: 37 seconds
luis@DESKTOP-LI6UPRI MINGW64 ~/appinventor-sources/appinventor (master)
```

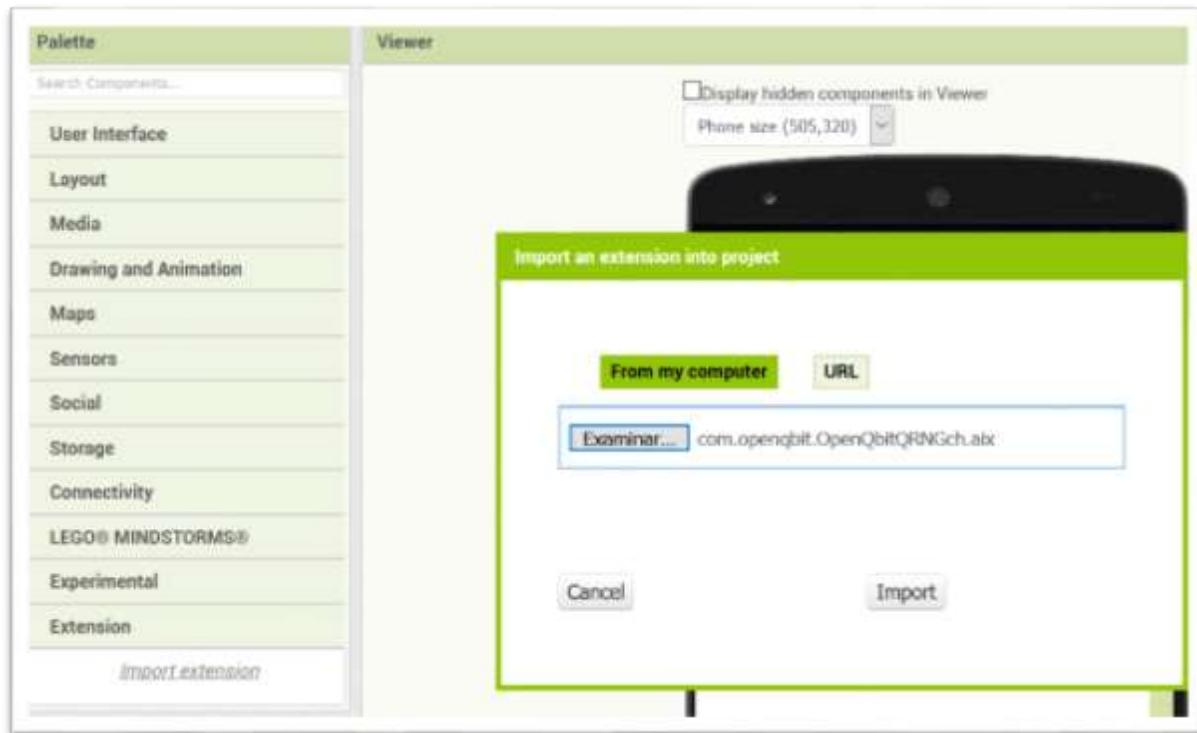
Vamos a essa pasta:

C:\Utentes\Luis-appinventor-fontes-pinvendor-componentes-construção de extensões

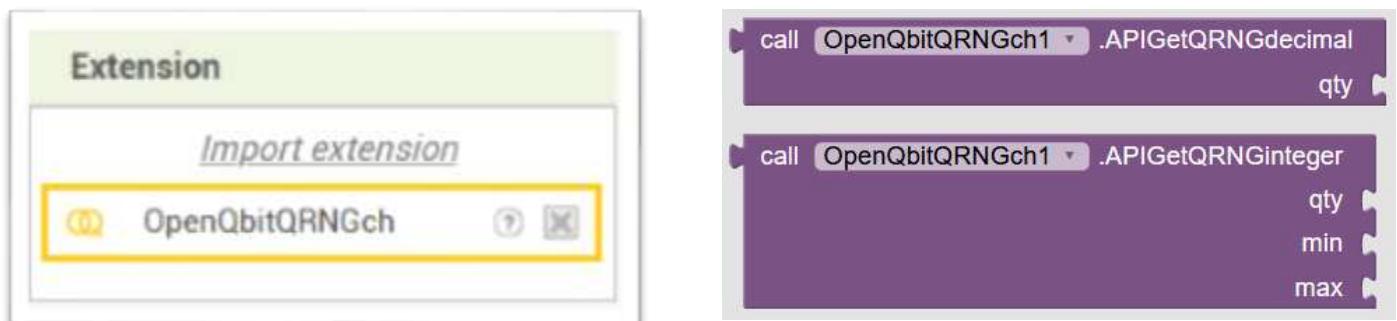
e copiar o ficheiro com.openqbit.OpenQbitQRNGch.aix, este ficheiro é a nossa extensão.

Tendo já uma conta no App Inventor ou outro sistema Blockly que introduzimos para adicionar a nova extensão e testá-la.

Ir para o fundo onde está a opção Extensão e clicar em "Importar extensão":



Carregamos no botão "Importar". Agora temos os blocos (**APIGetQRNGdecimal**) e (**APIGetQRNGinteger**) a serem utilizados:



Já temos a nossa primeira extensão criada e funcional.

27.Anexo "BlocklyCode Smart Contracts".

Os BlocklyCodes são programas feitos em linguagem java. A extensão para criar este tipo de programa comumente chamado "Smart Contracts" é uma forma de processar acordos entre utilizadores (empresas ou pessoas).

Este bloco (**BlocklyCode**), é implementado num programa que já tem parâmetros estabelecidos e que, dependendo do tipo de acordos que poderiam ser executados pelo sistema Mini BlocklyChain de forma automática quando as premissas que regem o "Contrato Inteligente" são cumpridas.

Os parâmetros a considerar sugeridos ou parâmetros de entrada 'inputs' são

Data de expiração

Data de referência

Tempo de expiração

Hora referenciada

Bem referenciado

Total de activos fixos

Activos variáveis

Membros do contrato

Dados confirmados (String)

Dados confirmados (Nome do ficheiro)

Evento variável

Evento fixo

Validação do(s) documento(s)

Validação de cordas

Assinatura válida

Intervalo definido (Inteiro)

Intervalo decimal

Mínimo estabelecido

Máximo estabelecido

Antes de começarmos a utilizar o bloco (**BlocklyCode**) devemos primeiro instalar o sistema que executará a execução dos "Contratos Inteligentes", isto é feito através da instalação no terminal do Termux OpenJDK e OpenJRE.

OpenJDK (Open Java Development Kit). - É a ferramenta para desenvolver programas em java, contém as bibliotecas, o compilador e a JVM (Java Virtual Machine).

OpenJRE (Open Java Runtime Environment). - Esta é a ferramenta para executar apenas programas java. Contém a JVM (Máquina Virtual Java).

Procedemos à instalação de OpenJRE e OpenJDK no terminal Termux dos nós que formam o sistema Mini BlocklyChain.

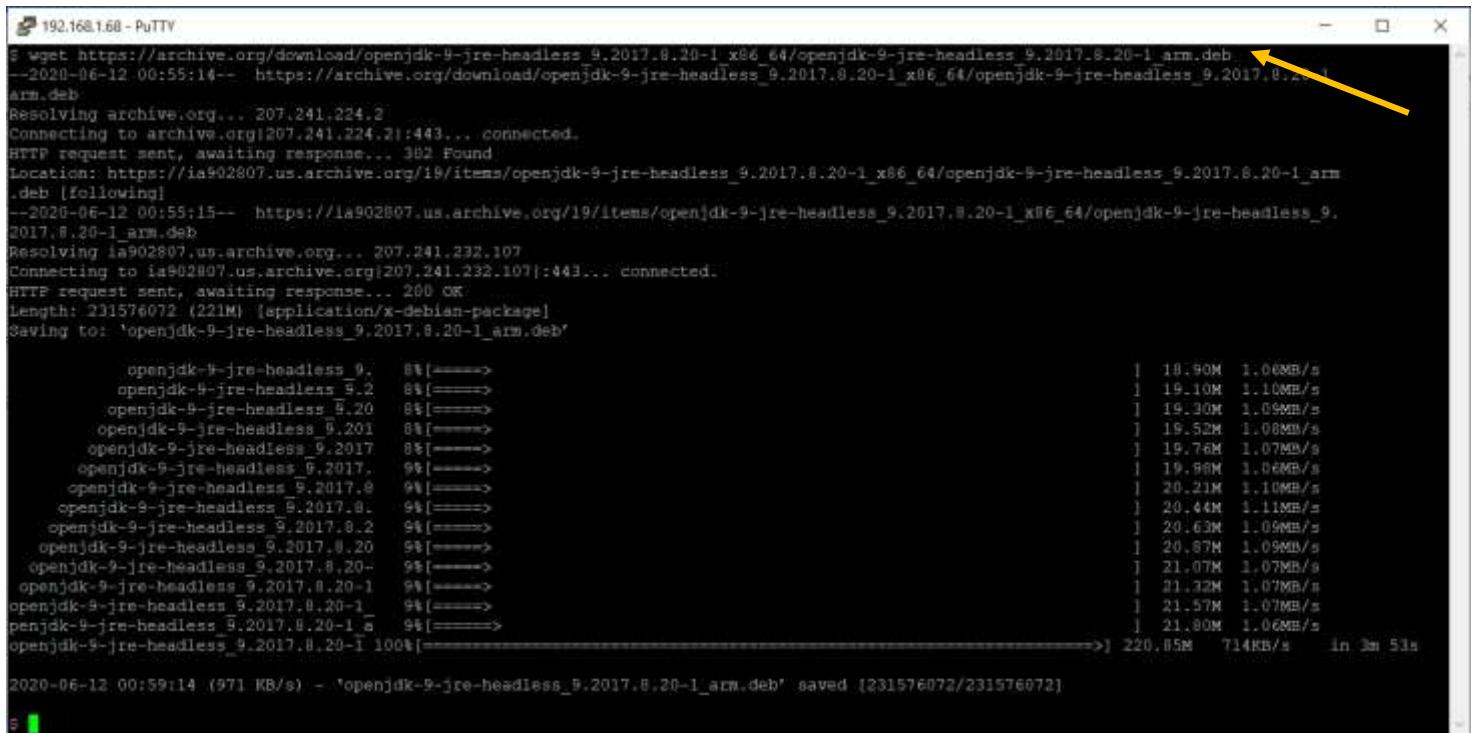
Instalamos os quartos

\$ apt install - e wget



Descarregámos o OpenJRE

```
$ wget https://archive.org/download/openjdk-9-jre-headless\_9.2017.8.20-1\_x86\_64/openjdk-9-jre-headless\_9.2017.8.20-1\_arm.deb
```

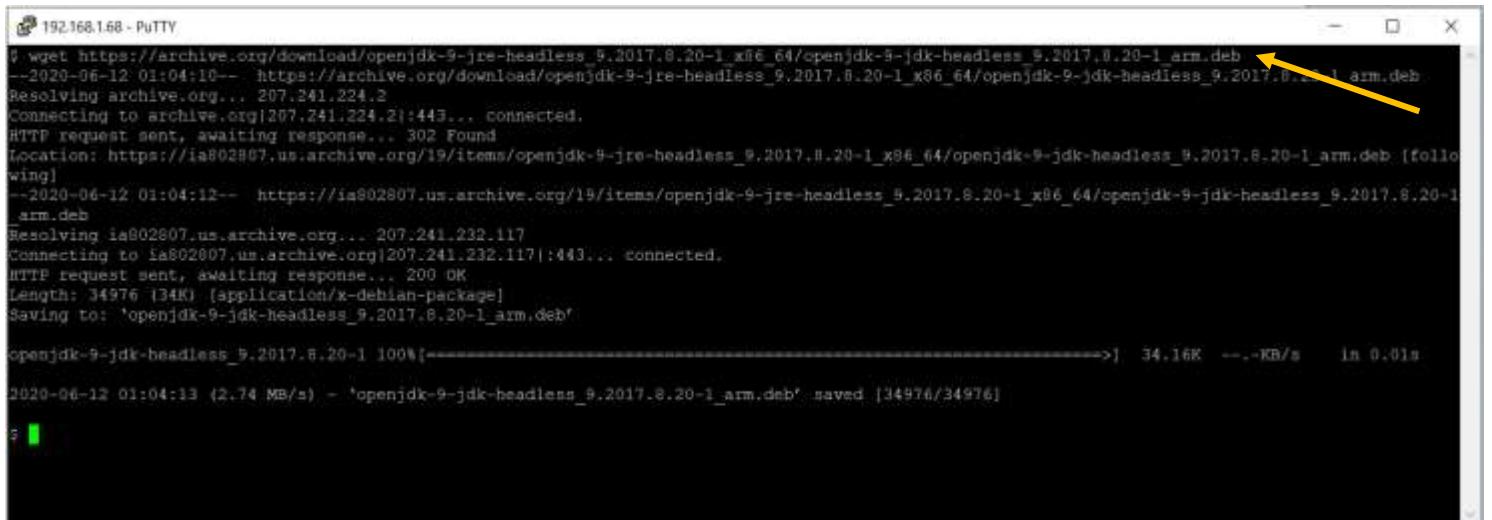


```
192.168.1.68 - PuTTY
$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
--2020-06-12 00:55:14-- https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Resolving archive.org... 207.241.224.2
Connecting to archive.org|207.241.224.2|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ia902807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb [following]
--2020-06-12 00:55:15-- https://ia902807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Resolving ia902807.us.archive.org... 207.241.232.107
Connecting to ia902807.us.archive.org|207.241.232.107|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 231576072 (221M) [application/x-debian-package]
Saving to: 'openjdk-9-jre-headless_9.2017.8.20-1_arm.deb'

openjdk-9-jre-headless_9.2017.8.20-1_arm.deb          100%[=====]  220.85M  714KB/s    in 3m 51s
2020-06-12 00:59:14 (971 KB/s) - 'openjdk-9-jre-headless_9.2017.8.20-1_arm.deb' saved [231576072/231576072]
```

Descarregámos o OpenJDK

```
$ wget https://archive.org/download/openjdk-9-jdk-headless\_9.2017.8.20-1\_x86\_64/openjdk-9-jdk-headless\_9.2017.8.20-1\_arm.deb
```



```
192.168.1.68 - PuTTY
$ wget https://archive.org/download/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
--2020-06-12 01:04:10-- https://archive.org/download/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
Resolving archive.org... 207.241.224.2
Connecting to archive.org|207.241.224.2|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ia902807.us.archive.org/19/items/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb [following]
--2020-06-12 01:04:12-- https://ia902807.us.archive.org/19/items/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
Resolving ia902807.us.archive.org... 207.241.232.117
Connecting to ia902807.us.archive.org|207.241.232.117|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 34976 (34K) [application/x-debian-package]
Saving to: 'openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb'

openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb          100%[=====]  34.16K  ---KB/s    in 0.01s
2020-06-12 01:04:13 (2.74 MB/s) - 'openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb' saved [34976/34976]
```

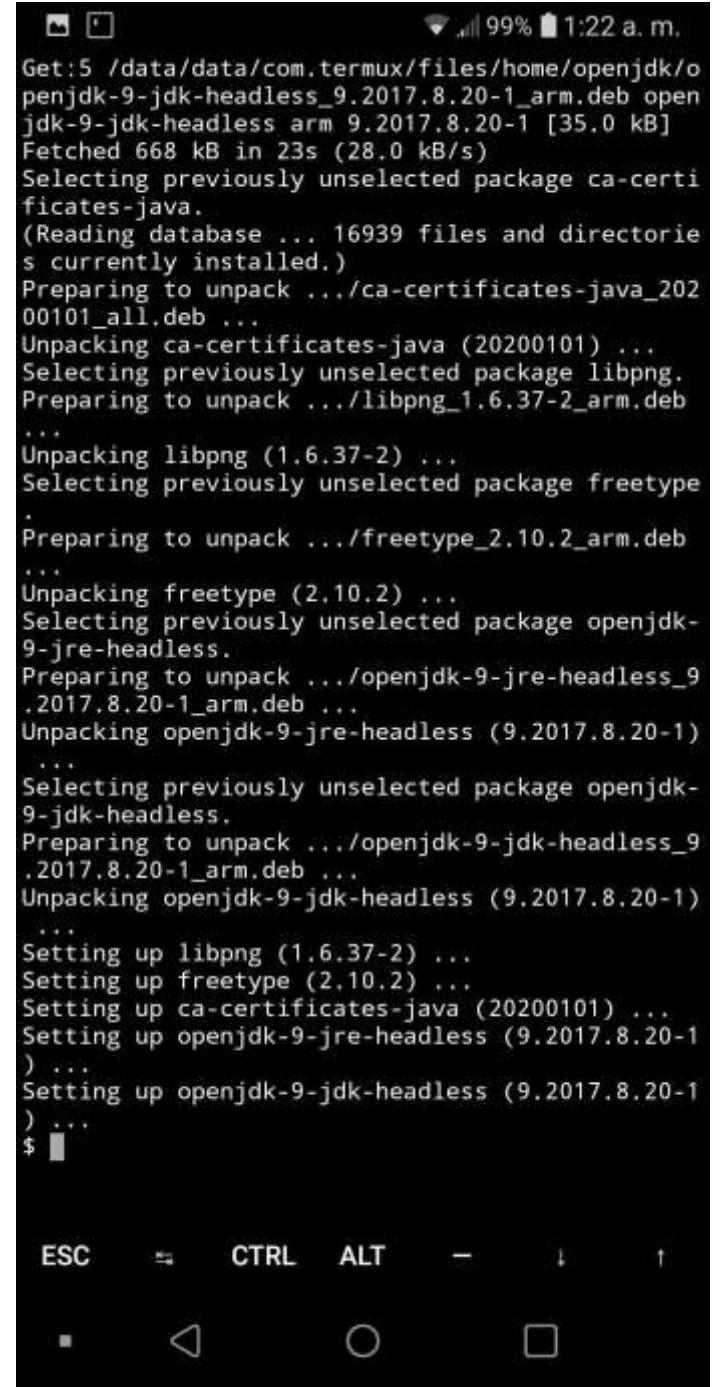
Efectuamos a instalação a partir do terminal Termux do OpenJDK e OpenJRE:

```
$ apt install -e ./openjdk-9-jre-headless_9.2017.8.20-1_arm.deb ./openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
```



```
$ ls
openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
$ apt install -y ./openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb ./openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'openjdk-9-jdk-headless' instead
of './openjdk-9-jdk-headless_9.2017.8.20-1_arm.
deb'
Note, selecting 'openjdk-9-jre-headless' instead
of './openjdk-9-jre-headless_9.2017.8.20-1_arm.
deb'
The following additional packages will be instal
led:
  ca-certificates-java freetype libpng
The following NEW packages will be installed:
  ca-certificates-java freetype libpng
  openjdk-9-jdk-headless
  openjdk-9-jre-headless
0 upgraded, 5 newly installed, 0 to remove and 0
not upgraded.
Need to get 668 kB/232 MB of archives.
After this operation, 376 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm ca-certificates-java all
20200101 [110 kB]
Get:2 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm libpng arm 1.6.37-2 [190
kB]
Get:3 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm freetype arm 2.10.2 [368
kB]
Get:4 /data/data/com.termux/files/home/openjdk/o
penjdk-9-jre-headless_9.2017.8.20-1_arm.deb open
jdk-9-jre-headless arm 9.2017.8.20-1 [232 MB]
Get:5 /data/data/com.termux/files/home/openjdk/o
penjdk-9-jdk-headless_9.2017.8.20-1_arm.deb open
jdk-9-jdk-headless arm 9.2017.8.20-1 [35.0 kB]

```



```
Get:5 /data/data/com.termux/files/home/openjdk/o
penjdk-9-jdk-headless_9.2017.8.20-1_arm.deb open
jdk-9-jdk-headless arm 9.2017.8.20-1 [35.0 kB]
Fetched 668 kB in 23s (28.0 kB/s)
Selecting previously unselected package ca-certi
ficates-java.
(Reading database ... 16939 files and directorie
s currently installed.)
Preparing to unpack .../ca-certificates-java_202
00101_all.deb ...
Unpacking ca-certificates-java (20200101) ...
Selecting previously unselected package libpng.
Preparing to unpack .../libpng_1.6.37-2_arm.deb
...
Unpacking libpng (1.6.37-2) ...
Selecting previously unselected package freetype
.
Preparing to unpack .../freetype_2.10.2_arm.deb
...
Unpacking freetype (2.10.2) ...
Selecting previously unselected package openjdk-
9-jre-headless.
Preparing to unpack .../openjdk-9-jre-headless_9
.2017.8.20-1_arm.deb ...
Unpacking openjdk-9-jre-headless (9.2017.8.20-1)
...
Selecting previously unselected package openjdk-
9-jdk-headless.
Preparing to unpack .../openjdk-9-jdk-headless_9
.2017.8.20-1_arm.deb ...
Unpacking openjdk-9-jdk-headless (9.2017.8.20-1)
...
Setting up libpng (1.6.37-2) ...
Setting up freetype (2.10.2) ...
Setting up ca-certificates-java (20200101) ...
Setting up openjdk-9-jre-headless (9.2017.8.20-1
) ...
Setting up openjdk-9-jdk-headless (9.2017.8.20-1
) ...
$ 
```

Desde que terminámos a instalação do ambiente OpenJDK e OpenJRE. Estas instalações contêm a JVM (Java Virtual Machine) que será o ambiente que irá executar o BlocklyCode.

Agora vamos instalar um editor para criar ficheiro online, vamos usar o editor chamado **vi** executar o seguinte comando:

```
$ apt install vim
```

Agora vamos tentar compilar um ficheiro de teste e executá-lo. Criamos no terminal Termux com o editor **vi** e escrevemos o código java de um "Hello World", e guardamo-lo no ficheiro HelloWorld.java

```
classe pública HelloWorld {  
  
    principal(String[] args) {  
        // Imprime "Olá, Mundo" para a janela terminal.  
        System.out.println("Olá, Mundo");  
    }  
  
}
```

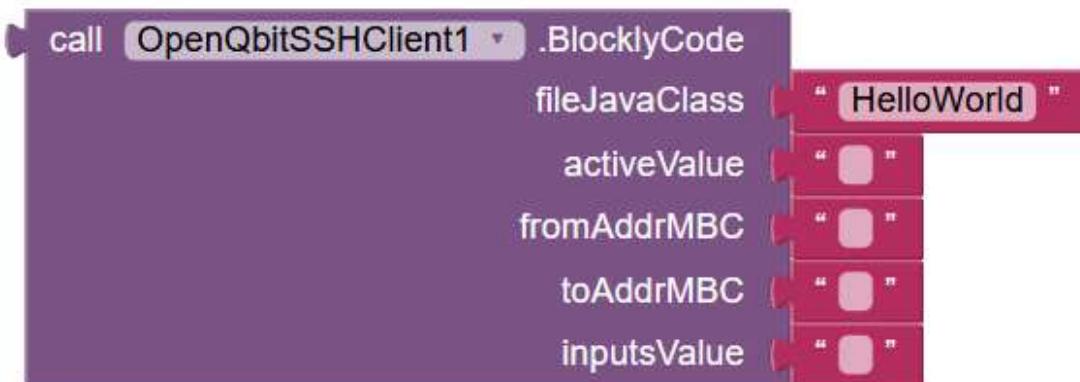
Em seguida, executamos o seguinte comando no terminal Termux para a compilação e depois a execução do programa:

```
$ javac HelloWorld.java (Depois de correr, é criado um ficheiro bytecode HelloWorld.class)
```

```
$ java HelloWorld (Depois de publicar e imprimir o anúncio, Hello World)
```



Usando o bloco (**BlocklyCode**) este bloco está localizado na extensão (**ConnectorSSHClient**).



No bloco anterior pode ver como será executado o ficheiro HelloWorld.class, que já foi compilado e posicionado no directório da base de utilizadores.

É um bloco no qual é possível executar um programa já compilado em java, que é comumente conhecido no ambiente de desenvolvimento como o ficheiro bytecode na sua forma binária.

Este tipo de ficheiro está pronto para ser executado pela Máquina Virtual Java (JVM).

Existem duas formas de criar um ficheiro bytecode ou com extensão .CLASS, o utilizador pode criá-lo no seu PC por comodidade ou semelhante também pode ser feito no telemóvel, lembre-se que já instalámos anteriormente o ambiente para compilar o JDK (Java Development Kit), embora seja menos eficiente porque a limitação do teclado contará, também no caso de escolher o ambiente Android terá de ter em conta que as bibliotecas estão limitadas ao OpenJDK que foi instalado.

Os parâmetros de entrada estão abertos em <inputsValue> e os outros parâmetros fixos são os de activeValue, fromaddrMBC e toAddrMBC.

No caso de testes de execução podem ser deixados em branco sem conteúdo e apenas o ficheiro bytecode será executado sem parâmetros.

O bloco anterior é como a próxima linha de comando a ser executada:

\$ java HelloWorld

Os programas desenvolvidos para BlocklyCode estarão sujeitos a cada ambiente de desenho particular e podem ser controlados e executados por rotina com o agente "cron" e partilhados com o syncthingmanager.

28.Anexo "OpenQbit Quantum Computing".

Como funciona a computação quântica? ⁽²⁾

A transformação digital está a provocar mudanças no mundo mais rapidamente do que nunca. Acreditaria que a era digital está prestes a terminar? A literacia digital já foi identificada como uma área onde o conhecimento aberto e as oportunidades acessíveis de aprender sobre a tecnologia são urgentes para colmatar lacunas no desenvolvimento social e económico. Aprender com os conceitos-chave da era digital tornar-se-á ainda mais crítico com a chegada iminente de outra nova onda tecnológica capaz de transformar os modelos existentes com velocidade e potência espantosas: **as tecnologias quânticas**.

Neste artigo, comparamos os conceitos básicos da computação tradicional e da computação quântica; e também começamos a explorar a sua aplicação em outras áreas relacionadas.

O que são tecnologias quânticas?

Ao longo da história, os seres humanos desenvolveram a tecnologia à medida que compreendiam como a natureza funciona através da ciência. Entre 1900 e 1930, o estudo de alguns fenómenos físicos que ainda não eram bem compreendidos deu origem a uma nova teoria física, a **Mecânica Quântica**. Esta teoria descreve e explica o funcionamento do mundo microscópico, o habitat natural das moléculas, átomos ou electrões. Graças a esta teoria, não só foi possível explicar estes fenómenos, como também foi possível compreender que a realidade subatómica funciona de uma forma completamente contra-intuitiva, quase mágica, e que no mundo microscópico ocorrem eventos que não ocorrem no mundo macroscópico.

Estas **propriedades quânticas** incluem sobreposição quântica, enredamento quântico e teleportação quântica.

- A **sobreposição quântica** descreve como uma partícula pode estar em diferentes estados ao mesmo tempo.
- O **enredamento quântico** descreve como duas partículas tão distantes quanto desejado podem ser correlacionadas de tal forma que, ao interagirem com uma, a outra está ciente disso.
- O **teletransporte quântico** utiliza emaranhamento quântico para enviar informação de um lugar para outro no espaço sem ter de viajar através dele.

As tecnologias quânticas baseiam-se nestas propriedades quânticas da natureza subatómica.

Neste caso, hoje em dia, a compreensão do mundo microscópico através da Mecânica Quântica permite-nos inventar e conceber tecnologias capazes de melhorar a vida das pessoas. Existem muitas e muito diferentes tecnologias que utilizam fenómenos quânticos e algumas delas, tais como lasers ou ressonância magnética (MRI), estão connosco há mais de

meio século. Contudo, estamos actualmente a assistir a uma revolução tecnológica em áreas como a computação quântica, informação quântica, simulação quântica, óptica quântica, metrologia quântica, relógios quânticos ou sensores quânticos.

O que é computação quântica? Primeiro, é preciso compreender a computação clássica.

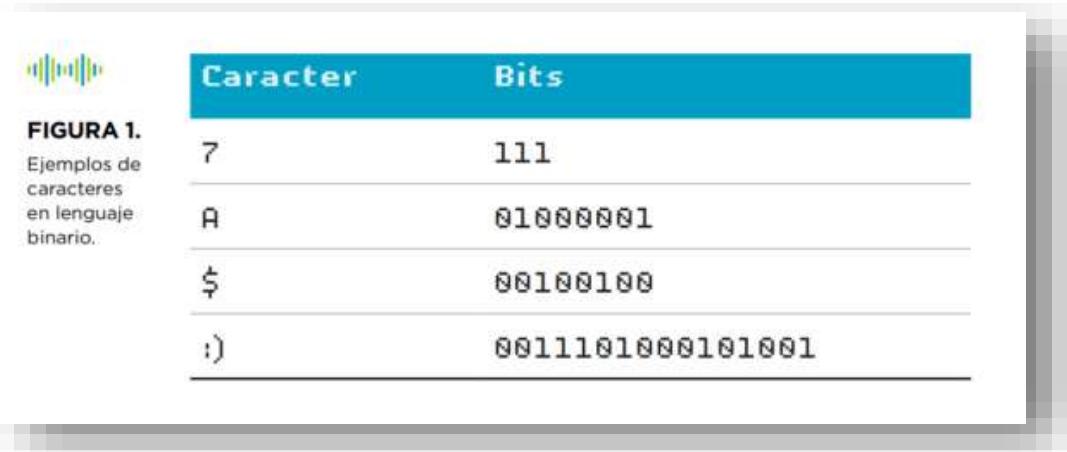


FIGURA 1. Ejemplos de caracteres en lenguaje binario.

Carácter	Bits
7	111
A	01000001
\$	00100100
:)	0011101000101001

Para compreender como funcionam os computadores quânticos, é conveniente explicar primeiro como funcionam os computadores que utilizamos todos os dias, que iremos referir neste documento como computadores digitais ou clássicos. Estes, tal como os restantes dispositivos electrónicos, tais como comprimidos ou telemóveis, utilizam bits como unidades fundamentais de memória. Isto significa que os programas e aplicações são codificados em bits, ou seja, em linguagem binária de zeros e uns. Sempre que interagimos com qualquer um destes dispositivos, por exemplo, premindo uma tecla no teclado, são criadas, destruídas e/ou modificadas cordas de zeros e uns dentro do computador.

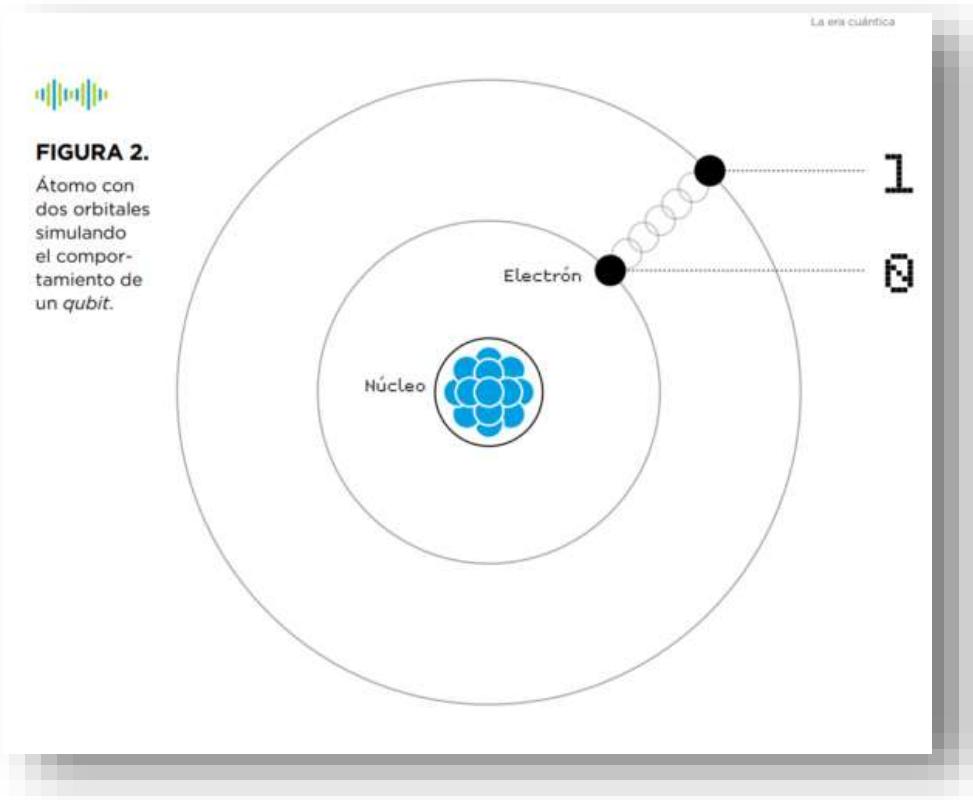
A questão interessante é, o que são estes zeros e uns fisicamente dentro do computador? O zero e um estados correspondem à corrente eléctrica que circula, ou não, através de peças microscópicas chamadas transístores, que actuam como interruptores. Quando não está a fluir corrente, o transistor está "desligado" e corresponde ao bit 0, e quando está a fluir está "ligado" e corresponde ao bit 1.

Mais simplesmente, é como se os bits 0 e 1 correspondessem a buracos, de modo que um buraco vazio é um bit 0 e um buraco ocupado por um electrão é um bit 1. Como exemplo, a figura 1 mostra a escrita binária de alguns caracteres. Agora que temos uma ideia de como funcionam os computadores de hoje, vamos tentar compreender como funcionam os quantum.

De bits a qubits

A unidade fundamental de informação na computação quântica é o bit quântico ou qubit. Os Qubits são, por definição, sistemas quânticos de dois níveis - veremos aqui exemplos - que,

tal como os bits, podem estar no nível baixo, que corresponde a um estado de baixa excitação ou energia definida como 0, ou no nível alto, que corresponde a um estado de maior excitação ou definido como 1. No entanto, e aqui reside a diferença fundamental com a computação clássica, as desistências podem também estar em qualquer dos infinitos estados intermédios entre 0 e 1, tais como um estado que é meio 0 e meio 1, ou três quartos de 0 e um quarto de 1.



Algoritmos quânticos, exponencialmente mais potentes e eficientes de computação

O objectivo dos computadores quânticos é tirar partido destas propriedades quânticas dos *qubits*, como sistemas quânticos que são, a fim de executar algoritmos quânticos que utilizam sobreposição e intercalação para fornecer um poder de processamento muito maior do que os clássicos. É importante salientar que a verdadeira mudança de paradigma não consiste em fazer o mesmo que os computadores digitais ou clássicos - os actuais - mas sim mais rápido, como se pode ler em muitos artigos, mas que os algoritmos quânticos permitem realizar certas operações de uma forma totalmente diferente que em muitos casos se revela mais eficiente - ou seja, em muito menos tempo ou utilizando muito menos recursos computacionais -.

Vejamos um exemplo concreto do que isto envolve. Imaginemos que estamos em Bogotá e queremos saber a melhor rota para chegar a Lima de entre um milhão de opções para lá chegar ($N=1,000,000$). A fim de utilizar computadores para encontrar o melhor caminho,

precisamos de digitalizar 1.000.000 opções, o que implica traduzi-las em linguagem bit para o computador clássico e em *qubits* para o computador quântico. Enquanto um computador clássico teria de ir um a um analisando todos os caminhos até encontrar o desejado, um computador quântico tira partido do processo conhecido como paralelismo quântico que lhe permite considerar todos os caminhos ao mesmo tempo. Isto implica que, enquanto o computador clássico necessita da ordem de passos $N/2$ ou iterações, ou seja, 500.000 tentativas, o computador quântico encontrará o caminho óptimo depois de apenas \sqrt{N} operações no registo, ou seja, 1.000 tentativas.

No caso anterior a vantagem é quadrática, mas noutras casos é mesmo exponencial, o que significa que com n *qubits* podemos obter uma capacidade computacional equivalente a 2^n bits. Para exemplificar isto, é comum contar que com cerca de 270 qubits poderíamos ter mais estados base num computador quântico - mais cadeias de caracteres diferentes e simultâneas - do que o número de átomos no universo, que é estimado em cerca de 280. Outro exemplo é que se estima que com um computador quântico entre 2000 e 2500 *qubits* poderíamos quebrar praticamente toda a criptografia utilizada hoje em dia (a chamada criptografia de chave pública).

Porque é importante saber sobre a tecnologia quântica?

Estamos num momento de transformação digital em que diferentes tecnologias emergentes, tais como cadeias de bloqueio, inteligência artificial, drones, Internet das coisas, realidade virtual, impressoras 5G, 3D, robôs ou veículos autónomos, têm cada vez mais presença em múltiplos campos e sectores. Estas tecnologias, chamadas a melhorar a qualidade de vida do ser humano acelerando o desenvolvimento e gerando impacto social, avançam hoje em dia de uma forma paralela. Só raramente vemos empresas a desenvolver produtos que exploram combinações de duas ou mais destas tecnologias, tais como a cadeia de bloqueios e a LPC ou os zangões e a inteligência artificial. Embora estejam destinados a convergir, gerando assim um impacto exponencialmente maior, a fase inicial de desenvolvimento em que se encontram e a escassez de promotores e pessoas com perfis técnicos significam que a convergência ainda é uma tarefa pendente.

Devido ao seu potencial disruptivo, espera-se que as tecnologias quânticas não só converjam com todas estas novas tecnologias, mas que tenham uma influência transversal em praticamente todas elas. A computação quântica ameaçará a autenticação, troca e armazenamento seguro de dados, tendo um grande impacto nas tecnologias em que a criptografia tem um papel mais relevante, como a segurança cibernética ou a cadeia de bloqueio, e um impacto negativo menor, mas também a ser considerado em tecnologias como a 5G, IoT ou drones.

Quer praticar computação quântica?

Dezenas de simuladores quânticos de computador já estão disponíveis na rede com diferentes linguagens de programação já em uso tais como C, C++, Java, Matlab, Maxima, OpenQbit.com

Python ou Octave. Também, novas línguas como o Q#, lançado pela Microsoft. Pode explorar e jogar com uma máquina quântica virtual através de plataformas como a IBM e a Rigetti.

Mini BlocklyChain é criado pela empresa OpenQbit.com que se concentra no desenvolvimento de tecnologia de computação quântica para diferentes tipos de sectores privados e públicos.

Porque é que a Mini BlocklyChain é diferente de outras cadeias de blocos, simplesmente porque o sistema foi criado para ser modular e incluir computação quântica.

- (2) <https://blogs.iadb.org/conocimiento-abierto/es/como-funciona-la-computacion-cuantica/>

29. Anexo "Blocos alargados para base de dados SQLite".

Para ver mais detalhes da utilização correcta e atempada de cada bloco que compõem a extensão OpenQbitSQLite verifique o autor original da extensão na seguinte ligação.

OpenQbitSQLite é um clone do desenho original com pequenas modificações para ser utilizado com um nível de segurança AES.

<https://github.com/frdfsnlght/aix-SQLite>

30. Anexo "Exemplo de criação do sistema Mini BlocklyChain".

Faremos um sistema de teste onde poderemos verificar as funcionalidades, vantagens e facilidade de criação de um sistema Mini BlocklyChain.

Começaremos com a concepção e desenvolvimento do armazenamento onde a informação residirá o que já definimos como uma cadeia de blocos. O desenho será baseado na base de dados SQLite e, dependendo de cada organização ou desenho, esta pode ser facilmente alterada por outra base de dados como Microsoft SQL Server, MySQL, Maria DB, Oracle, DB2, PostgreSQL entre outras. Embora mais uma vez devido ao processo de transacções e concorrência com a base de dados SQLite possa ser utilizada a qualquer nível e para aplicação comercial ou pessoal.

A criação de uma base de dados denominada **minibc** terá aqui uma tabela que armazenará a cadeia de blocos. Esta base de dados será integrada no código final do programa que será instalado nos nós, este local de armazenamento é chamado "assets packaged" é uma espécie interna em ambientes Blockly como o App Inventor.

\$ sqlite3 minibc.db

SQLite versão 3.32.2 2020-06-20 15:25:24

Introduzir ".help" para dicas de utilização.

Ligado a uma base de dados in-memory transitória.

Utilizar ".open FILENAME" para reabrir numa base de dados persistente.

sqlite> .quit

Desenho de campos da mesa de **nblock**.

CRIAR TABELA **nblock** (

id integer chave primária AUTOINCREMENTO

, prevhashVARCHAR NOT NULL

, newhashVARCHAR NOT NULL

ntransINTEGER NÃO NULL

nonceINTEGER NOT NULL

,qrng INTEIRO NÃO NULO

);

Criámos a mesa **nblock**.

\$ sqlite3 minibc.db

SQLite versão 3.32.2 2020-06-20 15:25:24

Introduzir ".help" para dicas de utilização.

Ligado a uma base de dados in-memory transitória.

Utilizar ".open FILENAME" para reabrir numa base de dados persistente.

sqlite> .open minibc.db

sqlite> CREATE TABLE nblock (id integer primary key AUTOINCREMENT , prevhash VARCHAR NOT NULL , newhash VARCHAR NOT NULL, ntrans INTEGER NOT NULL ,nonce INTEGER NOT NULL ,qrng INTEGER NOT NULL);

Veremos algo semelhante a:

```

$ sqlite3 minibc.db
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
sqlite> CREATE TABLE nblock (
...>     id integer primary key AUTOINCREMENT
...>     , prevhash VARCHAR NOT NULL
...>     , newhash VARCHAR NOT NULL
...>     , ntrans INTEGER NOT NULL
...>     , nonce INTEGER NOT NULL
...>     , qrng INTEGER NOT NULL);
sqlite> .tables
nblock
sqlite> .quit
$ 

```

Declaração SQL para a criação da tabela nblock.

Em seguida, vamos criar o hash inicial do sistema chamado "**Génesis**", que se baseia na criação de um novo hash do primeiro bloco da cadeia de blocos, vamos utilizar o bloco (**NewBlock**). Depois criaremos um segundo hash que chamaremos "um" com base no hash "Génesis" porque deve ser consistente com o primeiro hash porque o teste de validação de hash na cadeia de blocos inicial deve sempre obedecer a: prevhash = newhash.

NewBlock. Para criar o hash "**Génesis**", utilizaremos os parâmetros de entrada:



Parâmetro de entrada: **dados** <Calça>, **anteriorHash** <Calça>

Parâmetro de saída: Um SHA256 Hash.

Neste caso usaremos como "previousHash" inicial igual a "0" e no caso dos dados de entrada "dados" será a palavra "Genensis".

Isto dar-nos-á um hash calculado da combinação de ambos os dados:

SHA256 (Génesis0) = eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642

A string acima será inserida na tabela nblock, esta string deve ser encriptada para isso utilizamos a extensão (**OpenQbitEncDecData**).



Utilizaremos a extensão OpenQbitSSHClient para inserir os dados na base de dados de minibc.db, seria a declaração INSERT para a base de dados de minibc.db da tabela nblock.

```
sqlite3 keystore.db "inserir em nblock (prevhash, newhash, ntrans, nonce, qrng) valores ('0', 'eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642', '1', '0', '0');"
```

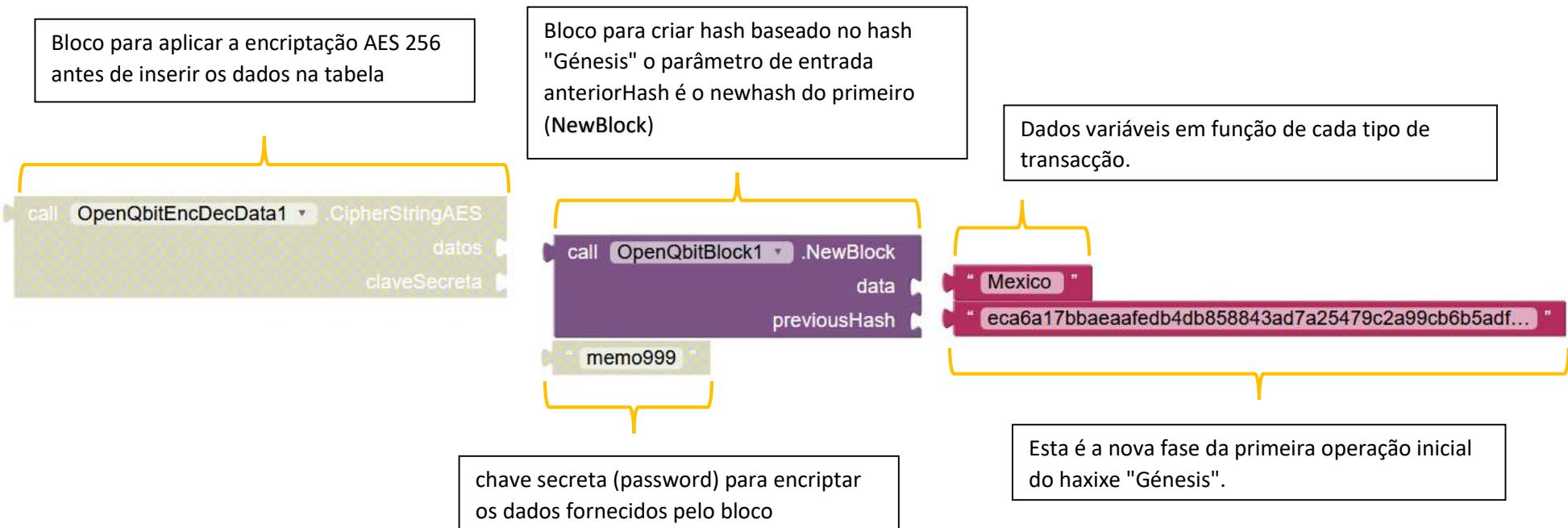
Criámos o hash "um" com a instrução SQL baseada no hash "Genensis":

```
sqlite3 keystore.db "inserir em nblock (prevhash, newhash, ntrans, nonce, qrng) valores ('eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642', 'f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68', '1', '0', '0');"
```

O hash "um" newhash é calculado como:

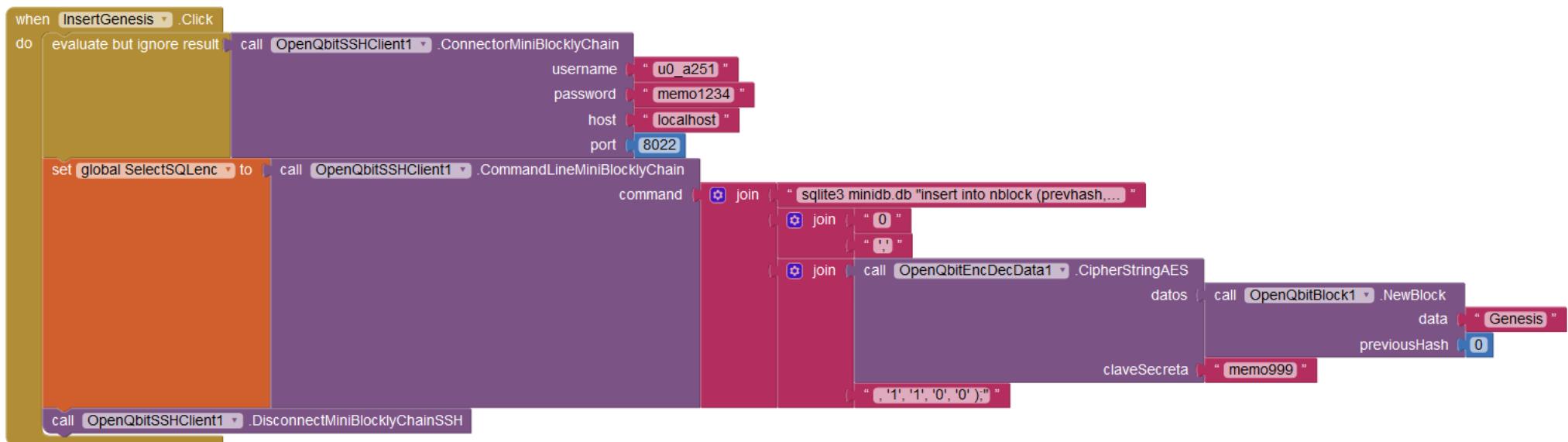
SHA256 (eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e802e642México) =
f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68

Segundo hash baseado no primeiro hash de inicialização "Génesis", devemos fazer dois INSERTS no início porque qualquer validação inicial da cadeia de blocos deve respeitar a igualdade de campos: prevhash (penúltimo dado) = newhash (último dado).



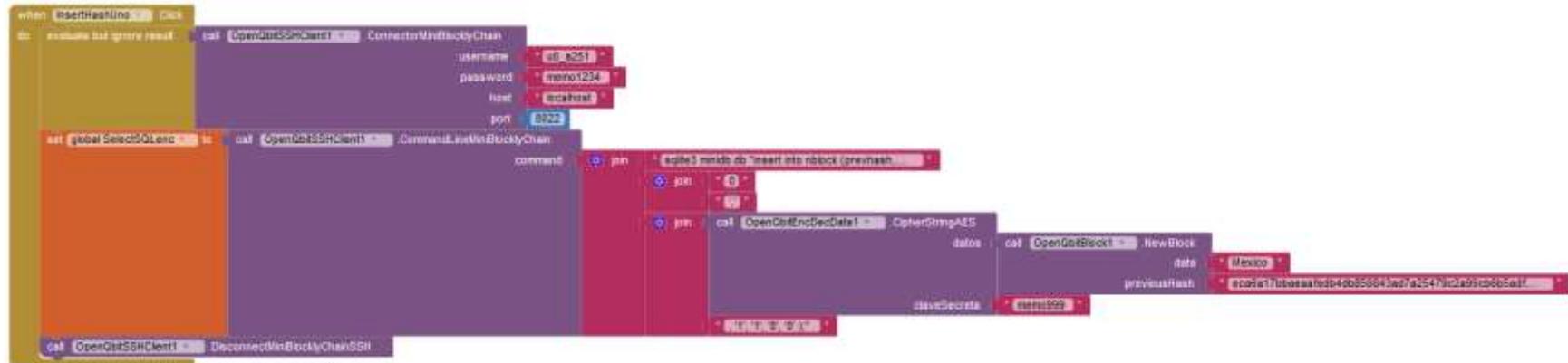
Continuamos com a criação da execução dentro do App Inventor dos hashes acima; hash "Génesis" e hash "start".

Agora mostramos como o INSERT do hash "Génesis" seria integrado na estrutura inicial da cadeia de blocos dentro da tabela nblock no sistema App Inventor:



Uma vez que inserimos os primeiros dados na tabela nblock com base no hash "Génesis", procedemos a fazer o segundo INSERT referenciado ao hash "um".

Integraria o INSERT do hash "Um" na estrutura inicial da cadeia de blocos dentro da tabela nblock no sistema App Inventor:



As duas estruturas de programação anteriores (hash "Génesis" e hash "um") devem ser integradas pelo nó inicial do sistema Mini BlocklyChain, um ponto importante é que a base de dados Mini BlocklyChain com toda a sua estrutura interna (tabelas) será replicada através da rede Mini BlocklyChain com base numa arquitectura "Peer to Peer".

Até agora, concluímos a estrutura básica e fundamental do armazenamento da cadeia de blocos e está pronto para receber novos blocos de futuras transacções originárias ou solicitadas no sistema.

Já inserimos os primeiros dados de hash "Génesis" e o hash "um" na nossa base de dados de **minibc.db**, agora vamos usar as seguintes instruções SQL para consultar os campos **prevhash** e **newhash** na tabela **nblock**.

Este ponto será fundamental, pois com base na comparação destes dois campos saberemos se a cadeia de blocos está a ser consistente e se mantém a integridade e a segurança das transacções. Este é apenas um mecanismo inicial para rever a cadeia de blocos, pois no futuro iremos rever a utilização do bloco para calcular o merkle tree, que será outra ferramenta essencial para garantir a integridade e a segurança da cadeia de blocos também no nosso sistema.

Por agora só iremos rever a estrutura ou sentenças SQL que devemos utilizar, tal como fizemos anteriormente com a extensão ([OpenQbitSSHClient](#)), com estas poderemos consultar os campos prevhash e newhash. Mais tarde podemos fazer a simples comparação de dados iguais para essa verificação cada vez que vamos adicionar um novo bloco.

Para prevhash:

```
sqlite3 minibc.db "SELECT prevhash FROM nblock ORDER BY id DESC LIMIT 1;"
```

Para newhash:

```
sqlite3 minibc.db "SELECT newhash FROM nblock ORDER BY id DESC LIMIT 1;"
```

Iremos agora centrar-nos na forma como faremos um pedido para submeter um novo comércio e/ou transacção a ser inserido na fila de transacções.

Há duas etapas como requisitos para enviar uma transacção para a fila de transacções.

O primeiro requisito é que o saldo da nossa conta tenha "activos" suficientes disponíveis para cobrir o montante a ser enviado. Lembre-se que os "bens" não podem ser apenas um número ou montante, mas podemos criar "bens" de qualquer tipo, dependendo de cada sistema a ser criado.

Exemplos de bens:

- ✓ Quantidade intrínseca de uma quantidade "virtual ou criptográfica" de uma nova origem.
- ✓ Dados referenciados a dados digitais (todos os tipos de documentos)
- ✓ Assinaturas de autenticação de hastes para cordas ou todos os tipos de ficheiros.
- ✓ Qualquer transacção ou transferência de informação digital ou representação da mesma.

O saldo de "bens" de cada nó é obtido utilizando o bloco (**GetBalance**).



O segundo requisito é fornecer os parâmetros mínimos aquando do envio de uma transacção, que são:

- ✓ Endereço de origem. - é o endereço a partir do qual os bens serão enviados.
- ✓ Endereço de destino. - é o endereço do utilizador que irá receber os bens enviados.
- ✓ Activo. - São os dados com valor intrínseco dado em cada sistema a serem enviados em cada transacção.

Os endereços acima (origem-destino) correspondem às chaves públicas de cada utilizador quando as contas de cada utilizador foram criadas. Lembre-se de que ao criar uma conta de utilizador, são criados dois tipos de chaves públicas e privadas.

A chave pública é o endereço que será partilhado em todo o sistema e que qualquer utilizador do sistema pode ver e utilizar para enviar ou receber transacções.

A chave privada é o endereço que é utilizado localmente por cada nó e como o seu nome indica, é para uso privado que ajuda a criar assinaturas digitais para dar segurança às transacções enviadas, esta chave nunca deve ser partilhada e é para uso local e única para o nó de origem.

A fim de utilizar os parâmetros anteriores, contamos com dois repositórios onde os endereços "chaves privadas" que se encontram na base de dados `keystore.db` são armazenados e serão de uso local de cada nó sem partilha no sistema e o outro repositório onde todos os endereços "chaves públicas" que se encontram na base de dados `publickeys.db` são armazenados serão partilhados através do protocolo "Peer to Peer" em todos os nós do sistema.

As bases de dados "`keystore.db`" e "`publickeys.db`" já foram criadas no Anexo "Criação de bases de dados KeyStore & PublicKeys".

A base de dados e o sistema para a fila de transacções já foi criada na secção "Instalação e configuração da rede RESTful Mini Sentinel". Onde já criámos uma base de dados para as transacções chamadas `op.sqlite3`, temos aqui duas tabelas, uma é "trans" que se ocupa das transacções (**fila de transacções**) e outra chamada "sign" onde são armazenadas as assinaturas digitais de cada operação.

O sistema SQLite RESTful é a rede de backup nesta ocasião, vamos usá-lo para facilitar e testar o sistema de backup, no entanto, para alterar e usar a mesma base de dados `op.sqlite3` num modelo "Peer to Peer" só teremos de usar a base de dados num modelo partilhado novamente no sistema de sincronização, o que veremos mais tarde.

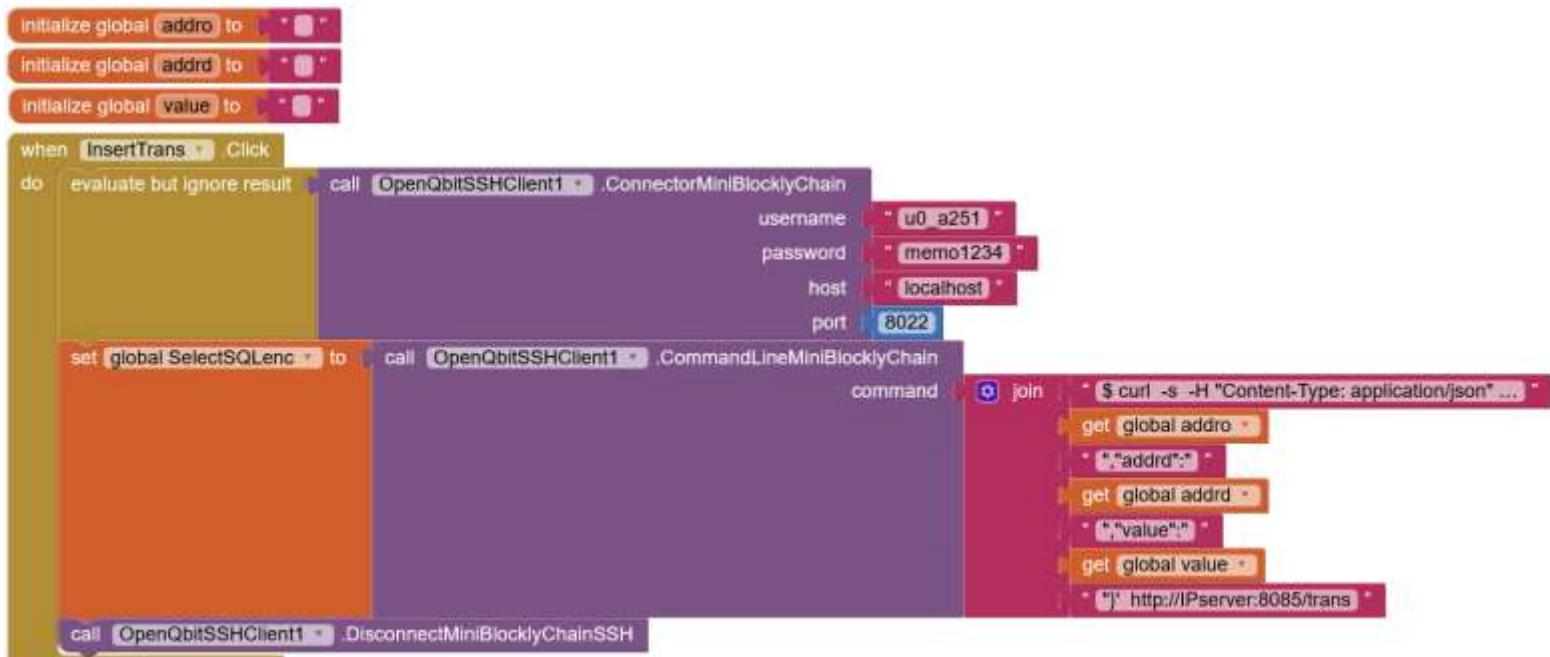
Começaremos a enviar operações para a fila de transacções utilizando RESTful aplicado à base de dados op.sqlite.

Criámos uma nova transacção na tabela "Trans".

```
$ curl -s -H "Content-Type: aplicação/json" -d
'{"addr": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",
"addrd": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh", "valor": "999"}'
http://IPserver:8085/trans
```

```
# resultados
{
  "id": 1,
  "addr": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",
  "addrd": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh",
  "valor": "999"
}
```

Implementação em App Inventor:



Os parâmetros de entrada: addro, addrd, valor são variáveis que podem ser introduzidas no algoritmo e são extraídas da base de dados keystore.db

Ver Apêndice "Criação de base de dados KeyStore".

Inserimos agora as assinaturas digitais da transacção anterior. Na tabela "sinal

```
$ curl -s -H "Content-Type: aplicação/json" -d '{  
  "trans_id":1  
  "sinal": "59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62",  
  "hash": " f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68"}'  
http://IPserver:8085/sign  
  
# resultados  
{  
  "id": 1,  
  "trans_id": 1,  
  "sinal": "59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62",  
  "hash": "f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68"  
}
```

NOTA: A assinatura digital (sinal) deve ser obtida antes de enviar a declaração de comando de ondulação, é gerada com o bloco (**GenerateSignature**).

Os parâmetros de entrada: Trans_id, sinal, hash são variáveis que podem ser introduzidas no algoritmo e a assinatura digital da transacção será gerada com o bloco (**GenerateSignature**).

Veremos agora o procedimento para gerar uma assinatura digital a ser aplicada a cada transacção que é feita.

Geração de assinatura digital para transacção. Quando utilizarmos o bloco (**GenerateSignature**) teremos como resultado um **ficheiro** do tipo **.sig** este ficheiro que será guardado no campo do sinal na tabela "sign, esta assinatura será utilizada quando a fila de transacção for processada e é outro parâmetro para dar segurança entre a origem e o destino, bem como a quantidade ou tipo de transacção entre eles.

Para tratar dados binários como o ficheiro.sig temos de convertê-lo para base64 e depois armazená-lo na variável sinal na tabela "sinal". Para tal, utilizaremos o bloco (**EncoderFileBase64**).

Como é calculado o valor do campo hash para a tabela de "sinal" de cada transacção:

Transacção Hash = SHA256(Endereço de origem + Endereço de destino + Activo + ficheiroBase64.sig)

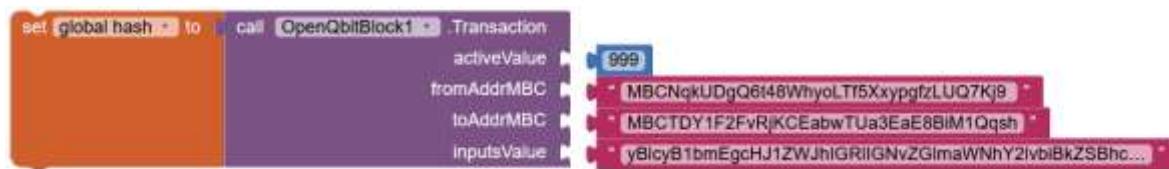
Exemplo de haxixe tipo SHA256:

SHA256(MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9 +

MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh + 999 +).

Saída: 9d45198faef624f2e7d1897dd9b3cede6ecca7fbac516ed1756b350fe1d56b4

Para o cálculo do hash teremos de nos apoiar no bloco (**Transacção**).



O parâmetro de saída é o hash que será armazenado para cada transacção que é enviada de qualquer nó do sistema. Este é armazenado no campo de sinal da tabela "sinal" na base op.sqlite

Com a operação anterior garantimos que apenas um hash único e irrepetível representará cada transacção enviada para a fila de espera e este hash representativo é a totalidade da informação transmitida.

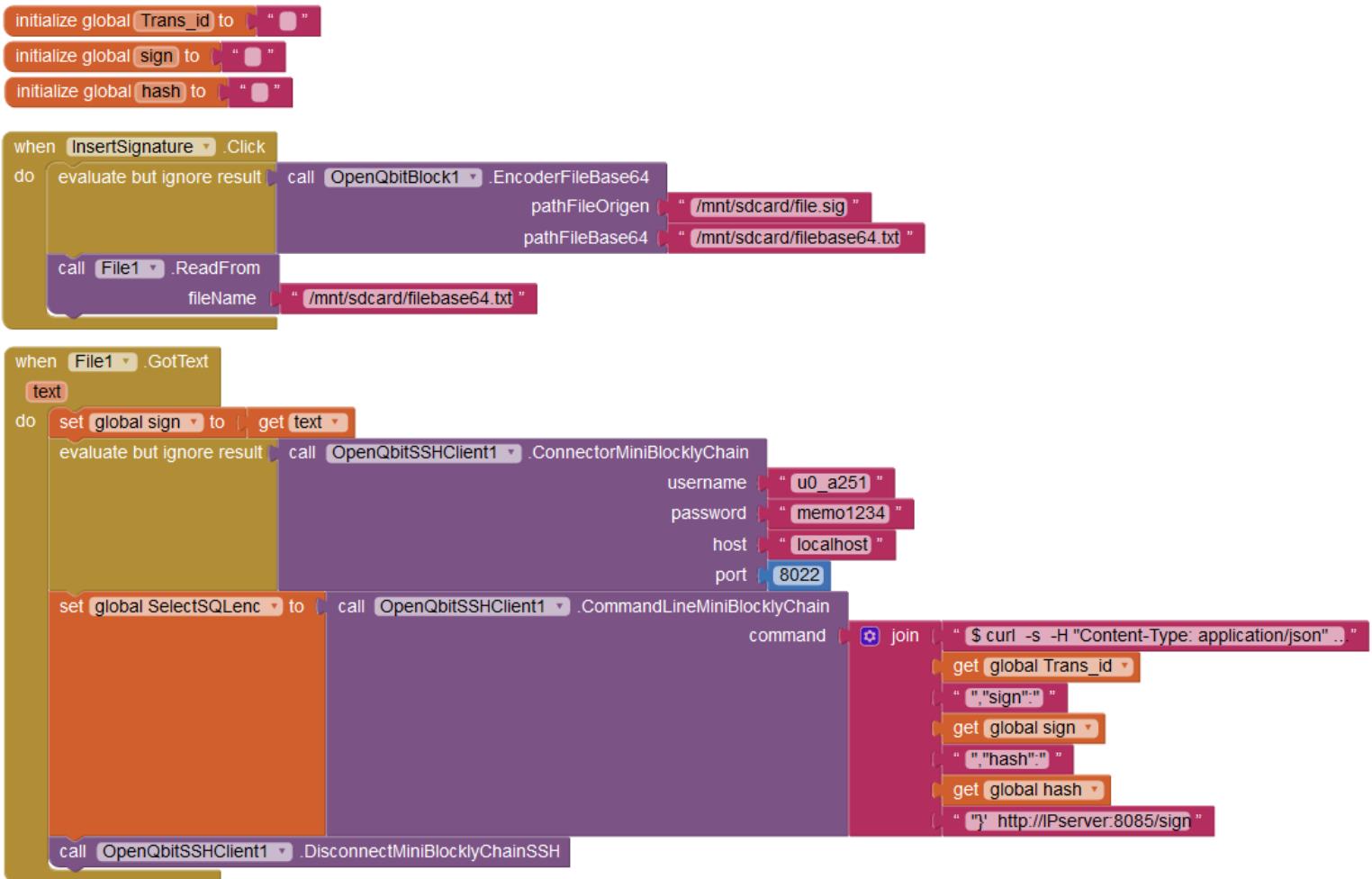
A única coisa que falta é incluir a variável Trans_id, que pode ser um identificador para diferenciar que nó é enviado para cada transacção. No nosso exemplo vamos colocar a variável Trans_id com um valor fixo de "1". Porque é o primeiro nó que está a ser configurado na nossa rede. Este valor pode variar a sintaxe de acordo com cada desenho em particular, por isso, para simplificar, escolhemos um identificador simples.

Uma vez que temos todas as variáveis definidas, vamos criar a estrutura e sequência de execução de blocos dentro do App Inventor, para realizar o INSERT na tabela "sinal".

NOTA: É importante ter em conta que cada envio de uma transacção é composto por dois INSERTS na base de dados op.sqlite3, um deve ser feito na tabela "trans" e os seus respectivos valores na tabela "sign".

Na concepção da base de dados op.sqlite3, foram criadas duas tabelas separadas devido ao facto de no futuro só ser possível encriptar a tabela "sinal", uma vez que contém informações que não devem ser enviadas na rede de forma plana, esta opção é deixada à consideração de cada concepção futura.

Vamos criar a estrutura de execução de blocos e métodos dentro do App Inventor do INSERT na tabela "sinal".



Até este momento, já terminámos o INSERT para a tabela "**sign**" e para a tabela "**trans**". Estas encontram-se dentro da base de dados **op.sqlite3** e os dados inseridos nestas duas tabelas serão utilizados para criar a fila de transacções que será enviada para todos os nós para o seu processamento.

Neste momento vamos utilizar o Conector Java SQLite-Redis. Este conector vai realizar a conversão dos dados da base de dados op.sqlite3 para a base de dados Redis. Ver Apêndice "Java SQLite-Redis Code Connector".

Após ter implementado o Conector SQLite-Redis, a fila de transacção será entregue a todos os nós do sistema através do sistema Redis.

Iniciaremos o processo de como podemos receber a fila de transacção de uma forma adequada para a podermos processar.

A fila de transacções será entregue através do serviço Redis. Esta é uma base de dados em tempo real que tem os seus respectivos blocos de controlo no ambiente App Inventor.

Configuração do ambiente Redis dentro do sistema Blockly do App Inventor.

Um ponto importante a salientar é que os blocos para controlar um servidor Redis até ao momento da redacção deste manual só estão disponíveis no sistema App Inventor. No caso de utilizar um sistema Blockly diferente do App Inventor, terá de utilizar a execução do Redis CLI (Command-Line) através do envio de comandos para o terminal Termux através da extensão ([OpenQbitSSHClient](#)).

Exemplo de utilizações em Redis CLI (Command-Line):

Para obter todas as **etiquetas** ou chaves na Redis.

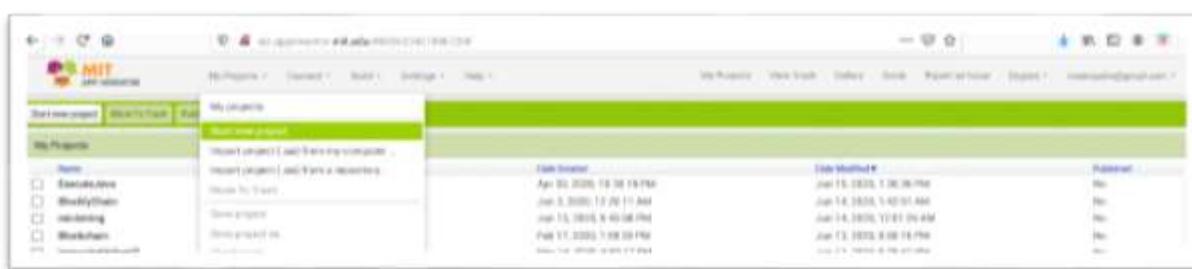
\$ redis-cli KEYS *

Para obter valor a partir de uma chave.

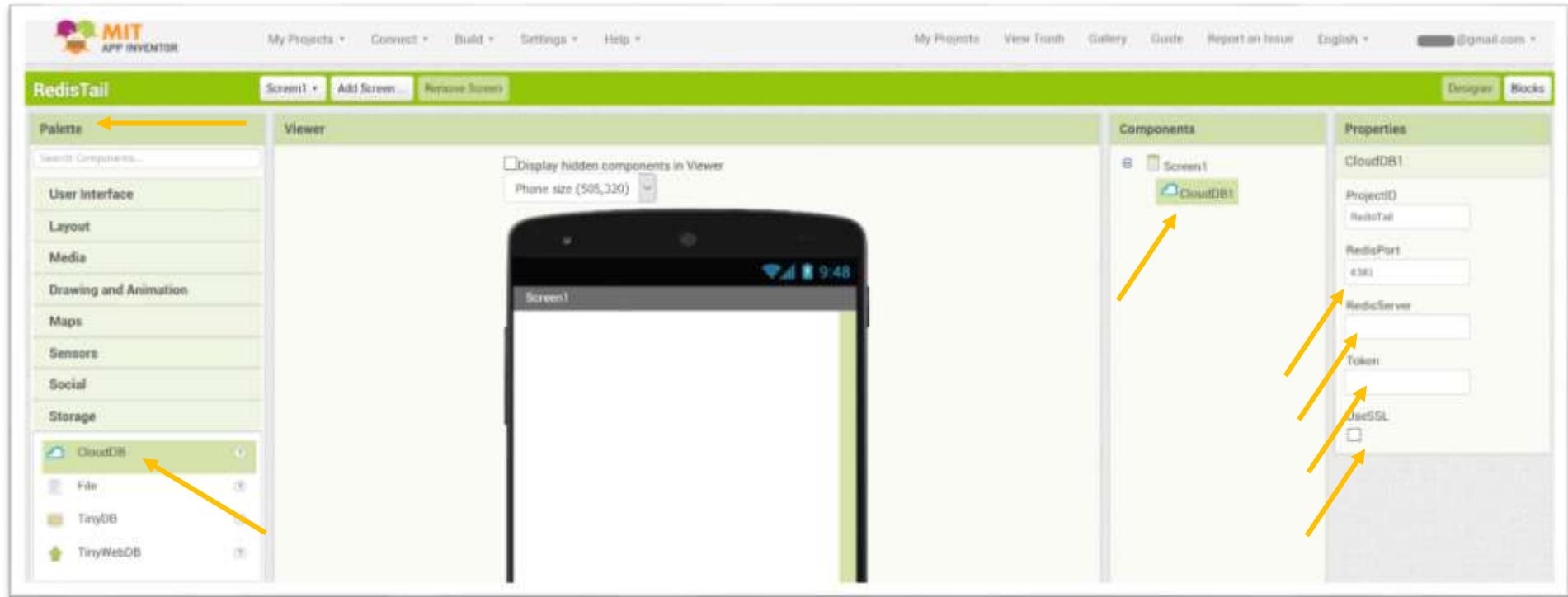
\$ redis-cli GET <key>

No nosso caso, porque estamos a utilizar o App inventor, iremos rever como utilizar os blocos para trabalhar com a Redis.

Dentro da App Inventor temos vários blocos para utilizar com a Redis, começámos a criar um novo projecto que iremos utilizar: Os meus projectos > Iniciar novo projecto



Depois de criarmos o projecto, vamos para o canto superior esquerdo e na secção "Paleta" clicar em "Armazenamento" e arrastar o objecto "CloudDB" isto integrará todas as funcionalidades para configurar uma ligação a um servidor Redis.



A configuração é muito simples, apenas temos de dar os seguintes parâmetros para podermos ligar ao nosso servidor Redis (Local) que está a funcionar no nosso nó (Telemóvel).

ProjectID. - Esta é a identificação que iniciará a etiqueta que identifica a fila de transacção na Redis.

RedisPort. - Esta é a porta do servidor Redis onde nos vamos ligar por defeito é 6379

RedisServer. - Este é o domínio ou IP local do nó no nosso caso e o de todos os nós será 127.0.0.1

Ficha. - Esta é a palavra-passe do servidor Redis activado para se ligar.

UseSSL. - Esta opção dá-nos a oportunidade de utilizar certificados SSL no nosso caso, deixando-o inactivo.

No nosso modelo e concepção do sistema teremos os seguintes parâmetros em todos os nós da rede.



É tempo de rever os blocos que nos fornecem controlo e processamento de dados sobre um ambiente de base de dados Redis.

Começamos com o bloco do método (**DataChanged**)



Este método dar-nos-á dois valores cada vez que houver uma mudança no servidor Redis que configurámos:

etiqueta. - Este valor é a etiqueta que identifica a fila de transacção.

valor. - Este valor contém a lista de todas as transacções enviadas para processamento. Este valor é uma lista de cordas do tipo <Corda>.

No caso do "valor" é onde começaremos a realizar o nosso processamento de informação da seguinte forma.

Ao fornecer-nos uma cadeia de todas as transacções de valor hash, começaremos por verificar se esta cadeia é válida e verificaremos se não foi modificada desde a sua origem. Fazemo-lo utilizando um algoritmo muito útil para a verificação de grandes quantidades de informação.

Utilizaremos o algoritmo de árvore de merkle. A aplicação deste algoritmo dá-nos uma segurança na integridade da informação que recebemos (fila de transacções).

Vejamos o seguinte exemplo de uma fila de transacção em Redis, executamos o Redis CLI (Command-Line) de um terminal Termux para verificar a chave "LATAM:HASH", devemos já ter executado o SQLite-Redis Connector para podermos consultar esta chave.

```
C:memor>redis-cli
127.0.0.1:6379> obter LATAM:HASH
"9d45198faaef624f2e7d1897dd9b3cde6ecca7fbac516ed1756b350fe1d56b4,"
f71c801a5fd25fc303ebc8c616204b4877ffb93006ec6a88bc30acf43ec250f5,"\60f8a3bcac1
ea7d38e86efbc3e3e00480807d23f980391000766e804ed14ecb2 ,
8a6dfe1d38c22e0f9212052efa6136da3edf1fb1b2a3e25224ac3d689124b754]
127.0.0.1:6379>
```

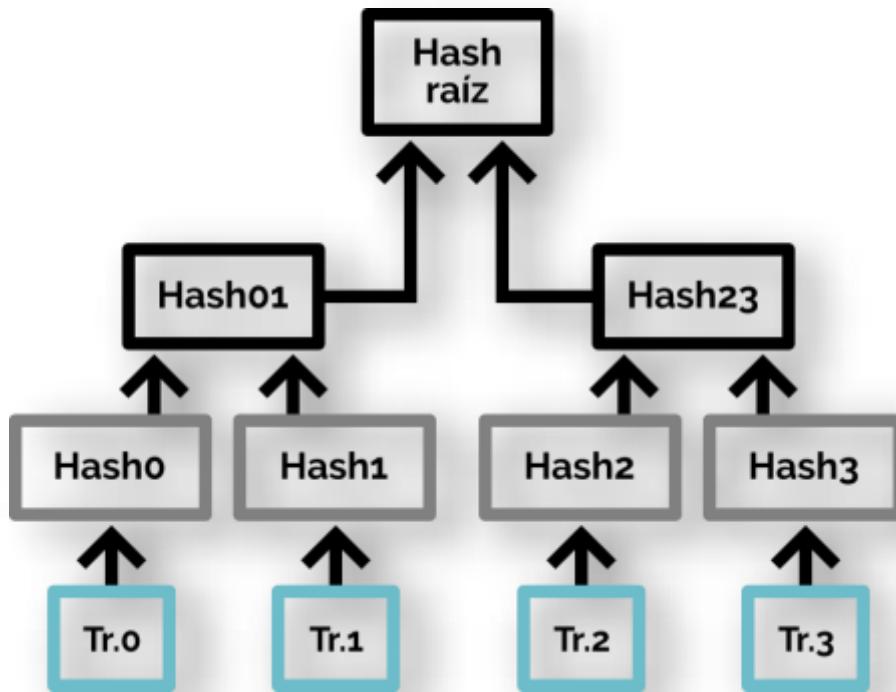
A fila de transacções anteriores tem apenas três elementos, uma vez que vemos quatro hashes representando uma transacção individual que a compõem e são os hashes de cada transacção que foi inicialmente injectada na base de dados op.sqlite3 dentro das tabelas "trans" e "sign".

Agora é tempo de compreender como funciona o merkle tree.

Uma árvore de hash merkle é uma estrutura de árvore de dados binária ou não binária na qual cada nó que não é uma folha é marcado com o hash da concatenação das etiquetas ou valores dos seus nós filhos. São uma generalização de listas de hash e cadeias de hash.

No nosso caso, faremos o seguinte cálculo para calcular o merkle tree para quatro elementos, isto é feito calculando o resultado de tomar pares de dados concatenados e obter os seus respectivos hashes, os resultados do primeiro nível serão aplicados aos resultados do segundo nível até haver apenas um elemento final, este será chamado o hash merkleroot (hash raiz).

Vejamos o seguinte diagrama que descreve este processo.



A fila de transacções baseada em hash será retirada através do bloco (`GetMerkleRoot`)



Raiz de haxixe:
51431822de7c94b90dc06d47b8f6275f315a4976c8479d30c32747fa90325432

O resultado é então comparado com a chave LATAM:merkleroot no sistema Redis local e ambas as chaves devem corresponder para verificar a integridade dos dados.

Outro ponto de segurança fornecido pelo merkle é confirmar que uma transacção específica está incluída na fila de transacção desde a sua origem e que não foi introduzida por nenhum meio de comunicação externo ou interno de forma fraudulenta.

No caso da cadeia ser de elementos estranhos na sua soma, o último elemento duplica para ter um arranjo e iniciar a execução do algoritmo.

Outro ponto não menos importante é o tempo para validar que cada transacção corresponde à sua origem-destino e que o bem é aquele que é enviado pelo endereço de origem.

É aqui que se encontra o Bloco (VerifySignature).

call OpenQbitBlock1 .VerifySignature

Antes de executar o bloco anterior, deve primeiro descarregar o ficheiro (file.sig) em formato binário a partir da tabela "sign":

```
sqlite3 op.sqlite3 "select sign from sign where=id_addr;"
```

id_addr: Esta é a identificação do endereço de origem da tabela "trans".

O pedido de pesquisa anterior dar-nos-á dados em formato Base64 da assinatura digital da transacção actual, isto para a converter no seu formato original (binário) precisaremos do Block(**DecoderFileBase64**).

Uma vez que temos o nosso ficheiro binário (file.sig) teremos de carregar no sistema a chave pública da origem e a chave pública do destinatário também em formato binário, tendo os quatro dados nos seus respectivos formatos será o momento de executar a verificação da assinatura digital no sistema.

- ✓ Endereço postal de chave pública
- ✓ Endereço de chave pública do destinatário
- ✓ Envio activo.
- ✓ Assinatura digital (file.sig)

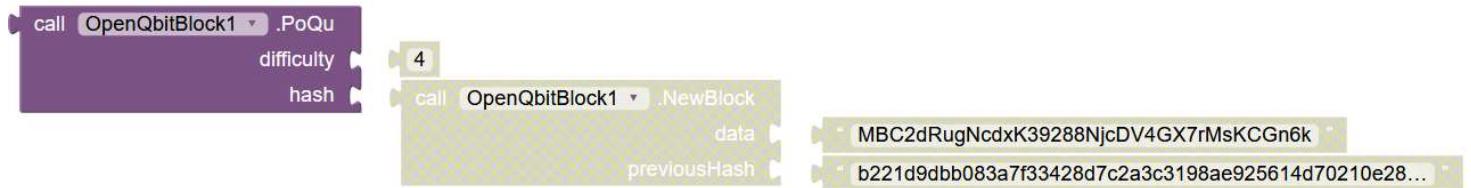
As chaves públicas em formato binário podem ser descarregadas no formato apropriado (binário) a partir da base de dados partilhada publickeys.db

Este processo deve ser executado para cada operação individual na fila de transacções.

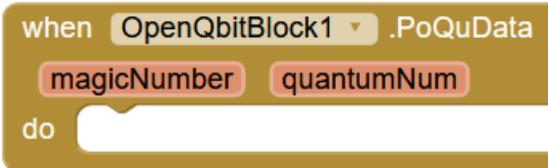
Finalmente, iremos rever como o sistema escolhe um nó de uma forma consensual para poder ser escolhido para adicionar o próximo bloco à cadeia de blocos e ser aquele que processa a fila de transacções.

Esta forma de escolher o nó vencedor para processar a fila de transacção baseia-se no nosso algoritmo desenvolvido para um sistema Mini BlocklyChain.

Como implementamos o consenso PoQu "Proof of Quantum". Este processo de consenso é baseado na geração de números quânticos aleatórios e é aplicado utilizando o bloco (**PoQu**).



Primeiro obtemos dois parâmetros que o bloco (**PoQu**) nos dá no método (**PoQuData**) os parâmetros são o **magicNumber** e o **quantumNum**.



O **magicNumber** é o número "nonce", é um número inteiro que é obtido fazendo um PoW interno com dificuldade não superior a 5, este número é responsável por dar um primeiro requisito ao nó com o **magicNumber** o nó poderá fazer um requisito para obter um número do sistema de geração de números quânticos aleatórios que está no intervalo do 0 a 1, este número dará uma probabilidade estabelecida aleatoriamente para o nó em execução que será armazenado na tabela chamada "vote".

A tabela "voto" armazenará o **quantumNum** calculado por cada nó, este armazenamento será feito com um número de nós até um certo tempo estabelecido em cada desenho de sistema que se recomenda ter de entradas $((N/2) + 1)$ onde "N" é a quantidade de nós disponíveis no sistema e pode ser estabelecido ou controlado por uma acção na ferramenta de gestão de tarefas "cron" de cada nó.

Um ponto importante é que por este ponto já deve ter estabelecido uma sincronização local da hora de cada nó através do sistema automático do telemóvel no caso de utilizar a rede "**Peer to Peer**" na transmissão da fila de transacções.

A configuração do agente cron nos nós. Ver secção "Sincronização do tempo nos nós do sistema (Telemóvel) em minutos e segundos".

Neste exemplo, como estamos a utilizar a rede de comunicações de reserva, não precisamos da sincronização de minutos e segundos para os nós porque o nosso exemplo ocupa um

esquema "**Cliente-Servidor**" este tipo de comunicação está apenas no processo de transmissão da fila de transacções. Todos os outros processos entre nós são através de uma comunicação "**Par a Par**".

Agora vamos analisar a estrutura, concepção e criação da tabela de "votação" que será localizada dentro da base de dados chamada "quorum.db" que também iremos criar neste exemplo.

\$ sqlite3

Versão SQLite 3.32.2 2020-06-20 15:25:24

Introduzir ".help" para dicas de utilização.

Ligado a uma base de dados in-memory transitória.

Utilizar ".open FILENAME" para reabrir numa base de dados persistente.

sqlite> .open quorum.db

sqlite> CREATE TABLE vote (id integer primary key AUTOINCREMENT NOT NULL, node_imei VARCHAR NOT NULL, quantumNum INTEGER NOT NULL, nonce INTEGER NOT NULL);

sqlite> .quit

A criação da mesma tabela de **votação** é mostrada abaixo, mas é através da introdução da declaração SQL numa forma segmentada:

\$ sqlite3

Versão SQLite 3.32.2 2020-06-20 15:25:24

Introduzir ".help" para dicas de utilização.

Ligado a uma base de dados in-memory transitória.

Utilizar ".open FILENAME" para reabrir numa base de dados persistente.

sqlite> .open quorum.db

sqlite> CREATE TABLE vote (

...> id integer chave primária AUTOINCREMENTO

...> node_imei VARCHAR NOT NULL,

...> quantumNum INTEGER NÃO NULL,

...> nonce INTEGER NÃO NULL

...>);

sqlite> .tabelas

votação

sqlite> .quit

Veremos algo semelhante na criação da base "quorum.db" e na votação de mesa.



A screenshot of a mobile device's screen. At the top, there is a status bar with icons for signal strength, battery level (26%), and time (9:02 p.m.). Below the status bar is a terminal window with the following text:

```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open quorum.db
sqlite> CREATE TABLE vote (
...>   id INTEGER primary key AUTOINCREMENT,
...>   node_imei VARCHAR NOT NULL,
...>   quantumNum INTEGER NOT NULL,
...>   time INTEGER NOT NULL
...> );
sqlite> .tables
vote
sqlite> .quit
$
```

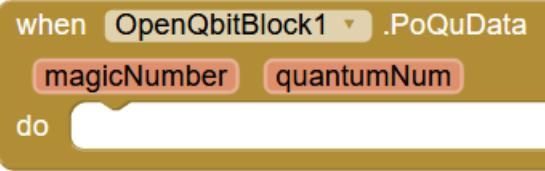
Below the terminal window is a virtual QWERTY keyboard. The keyboard layout includes numbers (1-0), letters (q-w-e-r-t-y-u-i-o-p, a-s-d-f-g-h-j-k-l-ñ), and various function keys like ESC, CTRL, ALT, and symbols.

Agora vamos ver como obtemos os valores da tabela de "votos".

nodo_imei. - **Obtemos** este valor utilizando o bloco (**GetDeviceID**).

call [OpenQbitBlock1] .GetDeviceID

quantumNum - Este valor é um dos resultados do método (**PoQuData**) que é obtido utilizando o bloco (**PoQu**).



nonce. - Este valor é obtido por já ter executado o bloco (**PoQu**) de forma integral e o mesmo que o número mágico do método (**PoQuData**).

Depois de completar o INSERTS na tabela "votação", é necessário fazer uma cópia da base de dados.

Após um certo tempo e com base na concepção de cada sistema, o serviço "cron" será executado em cada nó da rede e terá o próximo SELECT a ser processado na tabela "vote":

```
SELECT node_imei FROM vote WHERE magicNumber= (SELECT max(magicNumber) FROM vote);
```

O SELECT anterior devolve o resultado IMEI com maior probabilidade, agora cada nó que executa o SELECT fará uma comparação do seu IMEI com o resultado IMEI e só o nó que corresponder criará um ficheiro com o maior número de probabilidade que será replicado na rede "Peer to Peer" por um ficheiro com o formato IMEI.mbc que conterá o IMEI do nó vencedor.

O nó vencedor será capaz de começar a processar a fila de transacções. Com base em todos os blocos acima.

Dois pontos importantes são que, dependendo de cada criação de sistema, três processos terão de ser revistos e personalizados por cada projectista.

1.- Quando o nó vencedor começa a processar a fila de transacção, deve ser implementado um método ou processo onde se deve verificar se o nó vencedor está online e se a comunicação com a rede não foi perdida.

2.- Quando o processamento da fila de transacção começa, o nó vencedor deve iniciar duas bandeiras de controlo indicando o início do processamento e outra para confirmar que o processamento da fila de transacção foi concluído. Estas duas bandeiras devem ser partilhadas na rede com todos os nós, o que ajudará a localizar falhas de ligação ou falhas de processamento ou demasiado tempo de processamento.

3.- Em caso de falha de comunicação ou outro evento em que o nó vencedor não tenha sido capaz de processar a fila de transacção, deve ser escolhido o nó seguinte no intervalo de probabilidade imediata.

O ponto anterior pode ser controlado com um serviço que verifica que o nó vencedor está online e pode utilizar o serviço "cron", o guião deve ser desenvolvido para cada caso

concebido, no entanto, um exemplo genérico de um guião shell é mostrado abaixo para que possa ser modificado de acordo com as necessidades de cada sistema Mini Blocklychain.

```
#!/bin/bash
dir="/data/data/com.termux/files/home/Sync/imei";
se [ !"$(ls $directório)" ]
então
sqlite3 quorum.db "UPDATE vote SET magicNumber=0 WHERE magicNumber= (SELECT
max(magicNumber) FROM vote);"
senão

MAX_NUM=$(sqlite3 quorum.db "SELECT max(magicNumber) FROM vote;")
IMEI_quorum=$(sqlite3 quorum.db "SELECT node_id FROM vote WHERE=MAX_NUM")
IMEI_local=$(cat device_imei) // Use o bloco (GetDevice)
se [ IMEI_quorum -eq IMEI_local ]
então
    tocar $MAX_NUM > IMEI.mbc
fi
fi
saída
```

31. Anexo "Integração com ambientes Ethereum & Bitcoin".

Agora veremos como podemos integrar os dois sistemas de cadeia de bloqueio mais conhecidos a nível mundial especializados em criptoméniás como o Ethereum e o Bitcoin.

Comecemos com a instalação do software que nos ajudará a realizar todas as transacções possíveis no ambiente Ethereum.

O que é o Ethereum?

Ethereum é uma plataforma de código aberto, descentralizada ao contrário de outras cadeias de blocos, Ethereum pode fazer muito mais. É programável, o que significa que os programadores podem utilizá-lo para criar novos tipos de aplicações.

Estas aplicações descentralizadas (ou "dapps") obtêm os benefícios da criptomontagem e da tecnologia de cadeias de bloqueio. São fiáveis e previsíveis, o que significa que, uma vez "carregados" no Ethereum, funcionarão sempre dentro do prazo previsto. Podem controlar os activos digitais para criar novos tipos de aplicações financeiras. Podem ser descentralizados, o que significa que nenhuma entidade ou pessoa os controla.

Neste momento, milhares de criadores em todo o mundo estão a criar aplicações no Ethereum e a inventar novos tipos de aplicações, muitas das quais podem ser utilizadas hoje em dia:

- Carteiras de divisas criptográficas que lhe permitem fazer pagamentos baratos e instantâneos com ETH ou outros activos
- Aplicações financeiras que lhe permitem pedir emprestado, emprestar ou investir os seus activos digitais
- Mercados descentralizados, permitindo a troca de bens digitais, ou mesmo a troca de "previsões" sobre eventos do mundo real.
- Jogos em que tem bens no jogo e pode mesmo ganhar dinheiro real.
- Tem contratos inteligentes que são programas com acordos a serem executados quando as premissas com as quais foi elaborado ou criado são cumpridas.

Os contratos inteligentes partilham semelhanças com os **DApps** (aplicações descentralizadas), mas também os separam de algumas diferenças importantes.

Tal como os contratos inteligentes, um DApp é uma interface que liga um utilizador a um serviço de um fornecedor através de uma rede de pares descentralizada. Mas, enquanto os contratos inteligentes precisam de um número fixo de participantes para serem criados, os DApps não têm limite para o número de utilizadores. Além disso, não se limitam apenas a aplicações financeiras tais como contratos inteligentes: um DApp pode servir qualquer propósito em que se possa pensar.

No nosso caso utilizaremos a biblioteca chamada "Web3j" que é desenvolvida em Java, e que permite interagir com a cadeia de bloqueio Ethereum de uma forma simples e intuitiva.

Executar o seguinte comando para instalar "Web3j":

```
$ curl -L get.web3j.io | sh
```



A screenshot of a smartphone's terminal application. The screen shows the command `$ curl -L https://get.web3j.io | sh` being run. Below the command, there is a large amount of text representing the progress of the curl command, showing download speeds and file sizes. After the download progress, the terminal continues with the execution of the `sh` command, which installs the Web3j package. The text indicates the successful download of a 20572 byte file and provides instructions for how to use the installed package. The bottom of the screen shows a standard smartphone keyboard interface.



A screenshot of a smartphone's terminal application, showing the completion of the Web3j installation. The terminal displays a message indicating that Web3j was successfully installed. It provides instructions for how to use the package by sourcing the configuration file (`source $HOME/.web3j/source.sh`). It also notes that opening a new shell will automatically perform this action. For more information, it points to the documentation at `https://docs.web3j.io/command_line_tools/`. The terminal prompt ends with a dollar sign (\$). The bottom of the screen shows a standard smartphone keyboard interface.

Depois temos de criar a variável de ambiente **\$JAVA_HOME** com o caminho onde se encontra o executável "java", uma vez que a biblioteca irá pesquisar esta variável para poder ser executada com sucesso.

```
$ JAVA_HOME= /dados/dados/com.termux/files/usr/bin
```

Uma vez criada a variável, temos de ir ao directório onde a biblioteca "Web3j" foi instalada executando o seguinte comando, um ponto importante é que o directório é escondido depois de dr o comando de "cd" colocamos um ponto "." E depois o directório sem espaços, como se segue:

```
$ cd .web3j
```

Uma vez dentro, testamos se a biblioteca está a funcionar correctamente com o seguinte comando:

```
versão ./web3j
```

Resulta em algo muito semelhante a:

```

$ ls
source.sh  web3j  web3j-4.5.16
$ ./web3j version

Version: 4.5.16
Build timestamp: 2020-03-06 14:13:49.943 UTC
$ 
```

Mais tarde criaremos uma carteira para ser utilizada no ambiente da cadeia de bloqueio do Ethereum da seguinte forma:

\$./web3j criar carteira

O comando anterior dá-nos o endereço de ethereum:

4598fe2fd6afe2508f58343c7d42f2ab492edf34

```
$ ./web3j wallet create

Please enter a wallet file password:
Please re-enter the password:
Please enter a destination directory location [/data/data/com.termux/files/home/.ethereum/testnet/keystore]:
Wallet file UTC--2020-06-27T06-12-23.819752000Z--4598fe2fd6afe2508f58343c7d42f2ab492edf34.json successfully created in: /data/data/com.termux/files/home/.ethereum/testnet/keystore
$
```

O resultado é um ficheiro em formato JSON contendo o endereço e dados encriptados para serem utilizados mais tarde para gerar a chave privada para a conta que acabou de ser criada.

Este ficheiro do tipo JSON será criado num directório padrão chamado keystore.

A partir daqui já podemos realizar operações utilizando e/ou executando o comando Web3j.

Podemos fazê-lo utilizando a extensão (ConnectorSSHClient).

Para saber como utilizar os diferentes parâmetros e operações da biblioteca "Web3j", podemos ajudar-nos consultando a documentação no seu sítio oficial.

<https://docs.web3j.io/>

Para o ambiente Bitcoin, temos duas opções: utilizar o bloco () que gera a conta Bitcoin e as suas respectivas chaves públicas e privadas.

Podemos utilizar estas chaves para integrar no ambiente Bitcoin instalando a biblioteca java "Bitcoij".

\$ npm instalar bitcoinj



```
$ npm install bitcoinj
+ bitcoinj@0.0.0
added 1 package from 1 contributor and audited 2
09 packages in 20.528s
$
```

Para utilizar esta biblioteca contamos com o seu sitio oficial.

<https://bitcoinj.github.io/>

Uma segunda opção para integrar no ambiente da cadeia de bloqueio Bitcoin é instalar o pacote Bitcoind para a Termux, conforme mostrado abaixo.

\$ apt install bitcoin



```
$ pkg install bitcoin
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  bitcoin
0 upgraded, 1 newly installed, 0 to remove and 1
9 not upgraded.
Need to get 3601 kB of archives.
After this operation, 15.0 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm bitcoin arm 0.20.0 [3601
kB]
Fetched 3601 kB in 2s (1253 kB/s)
Selecting previously unselected package bitcoin.
(Reading database ... 19110 files and directorie
s currently installed.)
Preparing to unpack .../bitcoin_0.20.0_arm.deb .
.
Unpacking bitcoin (0.20.0) ...
Setting up bitcoin (0.20.0) ...
$
```

Agora, quando executarmos o agente bitcoind no terminal Termux, este criará automaticamente um endereço no directório:

/dados/dados/com.termux/files/hme/.bitcoin

Neste caso de Bitcoin, contamos com a seguinte documentação do agente "**Bitcoind**".

Neste caso também podemos utilizar a extensão (**ConnectorSSHClient**) para executar o agente "bitcoind", dependendo de cada necessidade.

Descrição dos parâmetros para utilização com bitcoind.

NOME

bitcoind - lançamento Bitcoin Core Daemon

SINOPSE

Bitcoind [*opções*] Iniciar *Bitcoin Core Daemon*

DESCRIÇÃO

Iniciar Bitcoin Core Daemon

OPÇÕES

-?

Imprimir esta mensagem de ajuda e sair

-alertnotify=<cmd>

Executar o comando quando é recebido um alerta relevante ou vemos um garfo muito comprido (%s em cmd é substituído pela mensagem)

-assumevalid=<hex>

Se este bloco estiver na corda assumir que ele e os seus antepassados são válidos e potencialmente saltar a sua verificação de escrita (0 para verificar tudo, por defeito:
000000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee, testnet:
00000000000037a8cd3e06cd5edbfe9dd1dbcc5dacab279376ef7cfxfc2b4c75)

-blocknotify=<cmd>

Executar o comando quando o melhor bloco muda (%s em cmd é substituído pelo hash do bloco)

-blockreconstructionextratxn=<n>

Transacções extra para guardar na memória para reconstruções de blocos compactos
(predefinição: 100)

-blocksdir=<dir>

Especificar o directório de blocos (por defeito: <datadir>/blocos)

-somente em bloco

Se recusar transacções de parceiros da rede. As transacções de carteira ou RPC não são afectadas. (predefinição: 0)

-conf=<archive>

Especificar o ficheiro de configuração. Os caminhos relativos serão prefixados pela localização do datadir. (predefinição: bitcoin.conf)

-daemon

Corre em segundo plano como um demónio e aceita os comandos

-datadir=<dir>

Especificar o directório de dados

-dbcache=<n>

Tamanho máximo da cache da base de dados <n> MiB (4 a 16384, por defeito: 450). Além disso, a memória mempool não utilizada é partilhada para esta cache (ver **-maxmempool**).

-debuglogfile=<file>

Especificar a localização do ficheiro de registo de depuração. Os caminhos relativos serão prefixados com uma localização de datadir específica da rede. (**-nodebuglogfile** para desactivar; por defeito: debug.log)

-includeconf=<file>

Especificar um ficheiro de configuração adicional, relativo ao caminho **-datadir** (só pode ser utilizado a partir do ficheiro de configuração, não a partir da linha de comando)

-loadblock=<file>

Importar blocos do ficheiro externo blk000???.dat no início

-maxmempool=<n>

Manter a reserva de memória de transacções abaixo de <n> megabytes (predefinição: 300)

-maxorphantx=<n>

Manter o máximo <n> transacções desconectáveis na memória (por defeito: 100)

-mempoolexpiry=<n>

Não manter transacções em mempool por mais de <n> horas (por defeito: 336)

-par=<n>

Define o número de fios de verificação no traço (-6 a 16, 0 = automático, <0 = deixa todos os núcleos livres, por defeito: 0)

-persistmempool

Se guardar o mempool quando desligar e carregá-lo quando reiniciar (predefinição: 1)

-pid=<file>

Especificar o ficheiro PID. Os caminhos relativos serão prefixados com uma localização de datadir específica da rede. (predefinição: bitcoind.pid)

-punza=<n>

Reducir as necessidades de armazenamento, permitindo a poda (remoção) de blocos antigos. Isto permite que a cadeia de poda RPC seja chamada para remover blocos específicos, e permite a poda automática de blocos antigos se for fornecido um tamanho alvo em MIB. Este modo é incompatível com **-txindex** e **-rescan**. Aviso: Para inverter esta definição é necessário voltar a descarregar toda a cadeia de blocos (por defeito: 0 = desactivar a poda de blocos, 1 = permitir a poda manual via RPC, >=550 = podar automaticamente os ficheiros de blocos para ficar abaixo do tamanho alvo especificado em MIB)

-reindex

Reconstrói o estado das cordas e o índice de blocos de ficheiros blk*.dat em disco

-reindex-chainstate

Reconstruir o estado da cadeia a partir dos blocos actualmente indexados. Quando em modo de poda ou se os blocos no disco puderem ser corrompidos, utilizar o **índice de re-indexação** completo.

-sigh

Criar novos ficheiros com permissões de sistema padrão, em vez de umask 077 (apenas eficaz com a funcionalidade de carteira desactivada)

-txindex

Manter um índice de transacção completo, utilizado pela chamada rpc da getrawtransaction (predefinição: 0)

-versão

Versão para imprimir e ir

Opções de ligação:

-addnode=<ip>

Adicionar um nó para ligar e tentar manter a ligação aberta (ver a ajuda do comando RPC da "adenda" para mais informações). Esta opção pode ser especificada várias vezes para adicionar vários nós.

-banscore=<n>

Limiar para desligar colegas mal comportados (por defeito: 100)

-Meanwhile...

Número de segundos para evitar que colegas mal-comportados se reconectem (por defeito: 86400)

-bind=<addr>

Amarre-se ao endereço dado e ouça-o sempre. Utilize a notação [host]:port notation for IPv6

-connection=<ip>

Ligar apenas ao nó especificado; **-noconectar desactiva** as ligações automáticas (as regras para este par são as mesmas que para o **-addnode**). Esta opção pode ser especificada várias vezes para se ligar a vários nós.

-descobrir

Descubra os seus próprios endereços IP (por defeito: 1 quando escuta e não **-externalip** ou **-proxy**)

-dns

Permitir pesquisas DNS para **-addnode**, **-seednode** e **-connect** (por defeito: 1)

-dnsseed

Consulta de endereços de pares através de pesquisa DNS, se os endereços estiverem baixos (por defeito: 1 a menos que **-ligação seja** utilizada)

-enablebip61

Enviar mensagens de rejeição por BIP61 (por defeito: 1)

-externalip=<ip>

Especifique o seu próprio endereço público

-forcednsseed

Consultar sempre os endereços dos colegas através da pesquisa no DNS (por defeito: 0)

-ouvir a

Aceitar ligações do exterior (por defeito: 1 se não houver **-proxy** ou **-conexão**)

-listenonion

Criar automaticamente o serviço oculto Tor (predefinido: 1)

-maxconnections=<n>

Manter no máximo <n> ligações com colegas (por defeito: 125)

-maxreceivebuffer=<n>

Máximo de buffer de recepção por ligação, <n>*1000 bytes (predefinição: 5000)

-maxsendbuffer=<n>

Buffer máximo de envio por ligação, <n>*1000 bytes (predefinição: 1000)

-configuração do tempo máximo

O máximo permitido para o ajustamento da compensação do tempo médio dos pares. A perspectiva do tempo local pode ser influenciada pelos pares para a frente ou para trás por esta quantidade. (por defeito: 4200 segundos)

-maxuploadtarget=<n>

Tente manter o tráfego de saída abaixo do alvo dado (em MiB durante 24h), 0 = sem limite (por defeito: 0)

-onion=<<ip:port>

Usar um proxy SOCKS5 separado para chegar aos pares através dos serviços ocultos do Tor, definir **-noonião** para desactivar (por defeito: **-proxy**)

-onlynet=<net>

Fazer ligações de saída apenas através da rede <net> (ipv4, ipv6 ou cebola). As ligações de entrada não são afectadas por esta opção. Esta opção pode ser especificada várias vezes para permitir múltiplas redes.

-filtros de pares

Suporta o bloqueio de filtragem e transacção com filtros de flores (predefinição: 1)

-permitbaremultisig

Relé não P2SH multisig (por defeito: 1)

-port=<port>

Oiça as ligações na "porta" (por defeito: 8333, testnet: 18333, regtest: 18444)

-proxy=<ip:port>

Ligar através do proxy SOCKS5, definir **-noproxy** para off (por defeito: off)

-proxirandomizar

Randomizar as credenciais para cada ligação por procuração Isto permite o isolamento do fluxo de Tor (por defeito: 1)

-seednode=<ip>

Ligar a um nó para recuperar os endereços do par, e desligar. Esta opção pode ser especificada várias vezes para se ligar a vários nós.

-timeout=<n>

Especificar o tempo limite de ligação em milissegundos (mínimo: 1, por defeito: 5000)

-torcontrol=<ip>:<port>

Porta de controlo de tor a utilizar se a escuta de cebola estiver activada (predefinição: 127.0.0.1:9051)

-password=<pass>

Palavra-passe da porta de controlo de tor (por defeito: em branco)

-upnp

Use UPnP para atribuir a porta de escuta (padrão: 0)

-whitebind=<addr>

Ligaçao a uma determinada morada e aos parceiros da lista branca que a ela se ligam.
Utilize a notação [host]:port notation for IPv6

-whitelist=<Endereço IP ou rede>

Os pares listados a branco são ligados a partir do endereço IP dado (por exemplo 1.2.3.4) ou da rede anotada do CIDR (por exemplo 1.2.3.0/24). Pode ser especificado várias vezes.
Os pares que constam da lista branca não podem ser proibidos pelo DoS

Opções de portfólio:

-tipo de morada

Que tipo de endereços utilizar ("legacy", "p2sh-segwit", ou "bech32", por defeito: "p2sh-segwit")

- evitar custos parciais

Agrupar as saídas por direcção, seleccionando todas ou nenhuma, em vez de seleccionar por cada saída. A privacidade é reforçada uma vez que um endereço só é utilizado uma vez (a menos que alguém o envie após ter sido gasto), mas pode resultar em taxas ligeiramente superiores, uma vez que a selecção de moedas pode ser subaproveitada devido à limitação acrescentada (por defeito: 0)

-mudança de tipo

Qual a taxa de câmbio a utilizar ("legacy", "p2sh-segwit", ou "bech32"). O padrão é o mesmo que **-addresstype**, excepto que **-addresstype=p2sh-segwit** utiliza saída segwit nativa quando envia para um endereço segwit nativo)

-da carteira

Não carregar a carteira e desactivar as chamadas RPC da carteira

-discardfee=<amt>

A taxa da tarifa (em BTC/kB) que indica a sua tolerância em rejeitar a alteração, adicionando-a à tarifa (por defeito: 0,0001). Nota: Uma saída é descartada se for poeira a esta taxa, mas descartamos sempre até à taxa de retransmissão de poeira e uma taxa de descargas acima dessa taxa é limitada pela estimativa da taxa para o alvo mais longo

-fallbackfee=<amt>

Uma taxa (em BTC/kB) a ser utilizada quando a estimativa da taxa tem dados insuficientes (por defeito: 0,0002)

-keypool=<n>

Definir o tamanho do grupo chave para <n> (por defeito: 1000)

-mintxfee=<amt>

Taxas (em BTC/kB) inferiores a esta são consideradas como taxa zero para a criação da transacção (por defeito: 0,00001)

-paytxfee=<amt>

Taxa (em BTC/kB) para adicionar às transacções que envia (por defeito: 0,00)

-rescan

Digitalizar novamente a cadeia de blocos para procurar transacções de carteira em falta no início

-salvagewallet

Tentativa de recuperar as chaves privadas de uma carteira corrompida no porta-bagagens

-desperdiçar a troca de informações

Gastar a alteração não confirmada ao enviar as transacções (por defeito: 1)

-txconfirmtarget=<n>

Se não for definida uma taxa de pagamento, incluir uma taxa suficiente para iniciar a confirmação das transacções em média dentro de n blocos (por defeito: 6)

-manutenção de carteiras

Actualizar a carteira para o formato mais recente no início

-wallet=<caminho>

Especificar o caminho da base de dados da carteira. Pode especificá-lo várias vezes para carregar várias carteiras. O caminho é interpretado em relação a <walletdir> se não for absoluto, e será criado se não existir (tal como um directório contendo um ficheiro de carteira.dat e ficheiros de registo). Para compatibilidade retroactiva, aceitará também os nomes dos ficheiros de dados existentes em <walletdir>).

-walletbroadcast

Fazer as transacções de difusão de carteira (por defeito: 1)

-walletdir=<dir>

Especificar o directório para guardar as carteiras (por defeito: <datadir>/portfolios se existir, caso contrário <datadir>)

-walletnotify=<cmd>

Executar comando quando uma transacção de carteira muda (%s em cmd é substituído por TxID)

-walletrbf

Enviar as transacções com a opção de incluir todo o RBF activado (RPC apenas, por defeito: 0)

-zapwallettxes=<mode>

Eliminar todas as transacções da carteira e recuperar apenas as partes da cadeia de bloqueio através de **-rescan** no início (1 = manter tx-metadata, por exemplo, informação de pedido de pagamento, 2 = largar tx-metadata)

Opções de notificação ZeroMQ:

-zmqpubhashblock=<endereço>

Habilitar o bloco de publicação em 'endereço'

-zmqpubhashblockhwm=<n>

Definir bloco de publicação de mensagens de saída com uma marca de água alta
(predefinição: 1000)

-zmqpubhashtx=<endereço>

Permitir a publicação da transacção de haxixe em 'endereço'

-zmqpubhashtxhwm=<n>

Definir publicar a mensagem de saída da transacção de hash de marca de água alta
(predefinição: 1000)

-zmqpubrawblock=<endereço>

Activar o bloco de publicação em bruto no 'endereço'

-zmqpubrawblockhwm=<n>

Definir Bloco Bruto Mensagem de Saída Marca de Água Alta (padrão: 1000)

-zmqpubrawtx=<endereço>

Permitir a publicação da transacção em bruto no "endereço"

-zmqpubrawtxhwm=<n>

Definir publicar mensagem de saída de transacção bruta marca de água alta (padrão:
1000)

Opções de depuração/teste:

-debug=<categoria>

Informação de depuração de saída (por defeito: **-nodebug**, desde que 'categoria' seja
opcional). Se <categoria> não for fornecida, ou se <categoria> = 1, toda a informação de
debug é emitida. <categoria> pode ser: net, tor, mempool, http, bench, zmq, db, rpc,
estimatefee, addrman, selectcoins, reindex, cmpctblock, rand, poda, proxy, mempoolrej,
libevent, coindb, qt, leveldb.

-debugexclude=<categoria>

Excluir a informação de depuração de uma categoria. Pode ser usado em conjunto com **-debug=1** para gerar registos de depuração para todas as categorias excepto uma ou mais categorias especificadas.

-help-debug

Imprimir mensagem de ajuda com opções de depuração e saída

-logips

Incluir os endereços IP na saída de depuração (padrão: 0)

-logtimestamps

Preparar a saída de depuração com o carimbo de tempo (por defeito: 1)

-maxtxfee=<amt>

Taxas máximas totais (em BTC) para utilização numa única transacção de carteira ou numa transacção bruta; se for definida demasiado baixa, pode abortar grandes transacções (por defeito: 0,10)

-impressão para a consola

Enviar informação de rastreio/debugging para a consola (por defeito: 1 quando não há **-daemon**. Para desactivar o registo para ficheiro, definir **-nodebuglogfile**)

-shrinkdebugfile

Reducir o ficheiro debug.log no arranque do cliente (por defeito: 1 quando não **-debug**)

-uacomment=<cmt>

Acrescentar um comentário à cadeia de agentes do utilizador

Opções de selecção em cadeia:

-testnet

Usar a cadeia de teste...

Opções de retransmissão dos nós:

-bytespersigop

Byte equivalentes por sigop em emissões e operações mineiras (por defeito: 20)

-datacarrier

Transacções de retransmissão e transporte de dados de minas (predefinição: 1)

-datacarrierize

Tamanho máximo de dados nas transacções dos suportes de dados que retransmitimos e extraímos (por defeito: 83)

-mempool-replacement

Permitir a substituição de transacções no pool de memória (predefinição: 1)

-minrelaytxfee=<amt>

Taxas (em BTC/kB) inferiores a esta são consideradas uma taxa zero para retransmissão, extracção e criação de transacções (por defeito: 0,00001)

-whitelistforcerelay

Forçar a retransmissão das transacções dos parceiros da lista branca, mesmo que as transacções já estivessem em mempool ou violassem a política local de retransmissão (padrão: 0)

-whitelistrelay

Aceitar transacções transmitidas recebidas de pares de listas brancas, mesmo quando não são transmitidas transacções (por defeito: 1)

Bloquear as opções de criação:

-blockmaxweight=<n>

Definir o peso máximo do bloco BIP141 (por defeito: 3996000)

-blockmintxfee=<amt>

Definir a taxa de comissão mais baixa (em BTC/kB) para transacções que estão incluídas na criação de blocos. (predefinição: 0,00001)

Opções de servidor RPC:

-Restaurant

Aceitar pedidos públicos de REST (predefinição: 0)

-rpcallowip=<ip>

Permitir ligações JSON-RPC a partir da fonte especificada. São válidos para <ip> um único IP (por exemplo, 1.2.3.4), uma rede/máscara de rede (por exemplo, 1.2.3.4/255.255.255.0), ou uma rede/CIDR (por exemplo, 1.2.3.4/24). Esta opção pode ser especificada várias vezes

-rpcauth=<userpw>

Nome de utilizador e palavra-chave HMAC-SHA-256 para ligações JSON-RPC. O campo 'userpw' vem no formato: 'nome de utilizador': 'SALT': \$ 'HASH'. Um guião de pitão canónico está incluído em share/rpcauth. O cliente liga-se então normalmente utilizando o rpcuser=<USERNAME>/rpcpassword=<PASSWORD> par de argumentos. Esta opção pode ser especificada várias vezes

-rpcbind=<addr>[:porto]

Faça a ligação a um determinado endereço para ouvir as ligações JSON-RPC. Não exponha o servidor RPC a redes não fiáveis, como a Internet pública! Esta opção é ignorada, a menos que **-rpcallowip** também seja passado. O porto é opcional e substitui o **-rpcport**. Utilize a notação [host]:port notation for IPv6. Esta opção pode ser especificada várias vezes (predefinição: 127.0.0.1 e ::1 i.e. localhost)

-rpccookiefile=<loc>

Localização do cookie de autorização. Os caminhos relativos serão prefixados por uma localização de datadir específica da rede. (default: data dir)

-rpcpassword=<pw>

Senha para ligações JSON-RPC

-rpcport=<port>

Oiça as ligações JSON-RPC em 'porto' (por defeito: 8332, testnet: 18332, regtest: 18443)

-rpcserialversion

Define a serialização da transacção em bruto ou o hexágono do bloco devolvido em modo não verbal, não segwit(0) ou segwit(1) (por defeito: 1)

-rpcthreads=<n>

Defina o número de fios para tratar as chamadas RPC (predefinição: 4)

-rpcuser=<utilizador>

Nome de utilizador para ligações JSON-RPC

-server

Aceitar comandos de linha de comando e JSON-RPC

32. Licenciamento e utilização de software.

Android

<https://source.android.com/setup/start/licenses>

Termux

<https://github.com/termux/termux-app/blob/master/LICENSE.md>

Nó

<https://raw.githubusercontent.com/nodejs/node/master/LICENSE>

SQLite

<https://www.sqlite.org/copyright.html>

Git

<https://git-scm.com/about/free-and-open-source>

sqlite-to-restante

<https://github.com/olsonpm/sqlite-to-rest/blob/dev/license.txt>

Redis DB

<https://redis.io/topics/license>

WorldTimeAPI NTP

<http://worldtimeapi.org/pages/faqs#commercial-apps>

Rede Tor

<https://github.com/torproject/tor/blob/master/LICENSE>

Rede de Sincthing

<https://forum.syncthing.net/t/syncthing-is-now-mpfv2-licensed/2133>

OpenSSH

<https://www.openssh.com/features.html>

Putty SSH

<https://www.chiark.greenend.org.uk/~sgtatham/putty/licence.html>

MIT App Inventor 2 Companheiro e App Inventor Blockly

<https://appinventor.mit.edu/about/termsofservice>

SQLite Expert Pessoal - freeware

<http://www.sqliteexpert.com/download.html>

Formiga Apache

<https://ant.apache.org/license.html>

MÍNE

<https://www.gnu.org/software/wget/>

OpenJDK

<https://openjdk.java.net/legal/>

Extensões externas:

JSONTOOLs

<https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html>

Licenciamento de versões opensource e comerciais do sistema Mini BlocklyChain consultar o site oficial <http://www.openqbit.com>

Mini BlocklyChain, MiniBlockly, BlocklyCode, MiniBlockMiniChain, QBlockly son marcas registradas por OpenQbit.

A Mini BlocklyChain é do domínio público.

Todo o código e documentação no Mini BlocklyChain foi dedicado ao domínio público pelos seus autores. Todos os autores de códigos e representantes das empresas para as quais trabalham assinaram declarações juramentadas dedicando as suas contribuições ao domínio público e os originais dessas declarações juramentadas são guardados num cofre nos escritórios principais da OpenQbit México. Qualquer pessoa é livre de publicar, usar ou distribuir as extensões originais do Mini BlocklyChain (OpenQbit), seja como código fonte ou como binários compilados, para qualquer finalidade, comercial ou não comercial, e por qualquer meio.

O parágrafo anterior aplica-se ao código e documentação entregável na Mini BlocklyChain, as partes da biblioteca da Mini BlocklyChain que realmente agrupam e enviam com uma aplicação maior. Alguns scripts utilizados como parte do processo de compilação (por exemplo, scripts de "configuração" gerados pelo autoconf) podem ser incluídos noutras licenças de código aberto. No entanto, nenhum destes guiões de compilação faz parte da biblioteca final da Mini BlocklyChain, pelo que as licenças associadas a esses guiões não devem ser um factor de avaliação dos seus direitos de cópia e utilização da biblioteca da Mini BlocklyChain.

Todo o código entregável em Mini BlocklyChain foi escrito do zero. Nenhum código foi retirado de outros projectos ou da Internet aberta. Cada linha de código pode ser rastreada até ao seu autor original, e todos esses autores têm dedicatórias de domínio público em ficheiro. Portanto, a base de código Mini BlocklyChain é limpa e não contaminada com código licenciado de outros projectos de código aberto, não contribuição aberta

A Mini BlocklyChain é de código aberto, o que significa que pode fazer quantas cópias quiser e fazer o que quiser com essas cópias, sem limitações. Mas a Mini BlocklyChain não é de código aberto. Para manter o Mini BlocklyChain no domínio público e para assegurar que o código não está contaminado com conteúdo proprietário ou licenciado, o projecto não aceita correcções de pessoas desconhecidas. Todo o código em Mini BlocklyChain é original, uma vez que foi escrito especificamente para utilização pela Mini BlocklyChain. Nenhum código foi copiado de fontes desconhecidas na Internet.

A Mini BlocklyChain é do domínio público e não necessita de licença. Mesmo assim, algumas organizações querem uma prova legal do seu direito a usar o Mini BlocklyChain. As circunstâncias em que isto ocorre incluem o seguinte:

- A sua empresa quer uma indemnização por queixas de violação de direitos de autor.
- Está a utilizar o Mini BlocklyChain numa jurisdição que não reconhece o domínio público.
- Está a utilizar o Mini BlocklyChain numa jurisdição que não reconhece o direito de um autor a colocar a sua obra no domínio público.
- Quer ter um documento legal tangível como prova de que tem o direito legal de usar e distribuir Mini BlocklyChain.
- O seu departamento jurídico diz-lhe que deve comprar uma licença.

Se alguma das circunstâncias acima se aplicar a si, a OpenQbit, a empresa que emprega todos os criadores da Mini BlocklyChain, vender-lhe-á uma Mini BlocklyChain Title Guarantee. Uma Garantia de Título é um documento legal que afirma que os autores reivindicados da Mini BlocklyChain são os verdadeiros autores, e que os autores têm o direito legal de dedicar a Mini BlocklyChain ao domínio público, e que a OpenQbit se defenderá vigorosamente contra as reivindicações de licenciamento. Todas as receitas da venda das garantias do título da Mini BlocklyChain são utilizadas para financiar a melhoria contínua e o apoio da Mini BlocklyChain.

Código Contribuído

Para manter o Mini BlocklyChain completamente livre e livre de royalties, o projecto não aceita remendos. Se quiser fazer uma sugestão de mudança e incluir um remendo como prova de conceito, isso seria óptimo. Contudo, não se ofenda se reescrevermos o seu remendo a partir do zero. O tipo de licença não comercial ou de fonte aberta que a utiliza nesta modalidade e algumas semelhantes, sem compra de apoio, quer individual quer colectiva, independentemente da dimensão da empresa, será regido pelas seguintes premissas legais.

Exoneração de responsabilidade pela garantia. A menos que exigido pela lei aplicável ou acordado por escrito, o Licenciador fornece a Obra (e cada Contribuinte fornece as suas Contribuições) "TAL COMO ESTÁ", **SEM GARANTIAS OU CONDIÇÕES DE QUALQUER TIPO, quer expressas ou implícitas, incluindo, sem limitação, quaisquer garantias ou condições de TÍTULO, NÃO-VIOLAÇÃO, COMERCIALIZAÇÃO OU ADEQUAÇÃO A UM FIM PARTICULAR.** É o único responsável por determinar a correcta utilização ou redistribuição da Obra e por assumir quaisquer riscos associados ao seu exercício das permissões ao abrigo desta Licença.

Quaisquer perdas financeiras ou outras incorridas pela utilização deste software serão suportadas pela parte afectada. Todas as disputas legais as partes submeter-se-ão aos tribunais apenas na jurisdição da Cidade do México, país México.

Para apoio comercial, utilização e licenciamento deve ser estabelecido um acordo ou contrato entre a OpenQbit ou a sua empresa e a parte interessada.

Os termos e condições do marketing de distribuição podem ser alterados sem aviso prévio, por favor ir ao site oficial www.openqbit.com para ver quaisquer alterações às cláusulas de apoio e licenciamento não-comerciais e comerciais.

Qualquer pessoa, utilizador, entidade privada ou pública de qualquer natureza legal ou de qualquer parte do mundo que simplesmente utilize o software aceita sem condições as cláusulas estabelecidas neste documento e aquelas que podem ser modificadas a qualquer momento no portal de www.openqbit.co sem aviso prévio e podem ser aplicadas à disposição do OpenQbit em uso não comercial ou comercial.

Quaisquer perguntas e informações sobre Mini BlocklyChain devem ser dirigidas à comunidade do App Inventor ou às várias comunidades do sistema Blockly, tal como estão: AppBuilder, Trunkable, etc. e/ou para o correio opensource@openqbit.com para a procura de perguntas pode levar a resposta de 3 a 5 dias úteis.

Apoio com uso comercial.

support@openqbit.com

Vendas para uso comercial.

sales@openqbit.com

Informação legal e questões ou preocupações sobre licenças

legal@openqbit.com

