



Установка, настройка и администрирование.

ИНСТРУКЦИЯ ПО ЭКСПЛУАТАЦИИ

версия 1.0 Beta

Июль 2020 года.

MiniBlocklyChain является зарегистрированной торговой маркой компании OpenQbit Inc., на условиях бесплатного использования и коммерческой лицензии. Условия использования: www.OpenQbit.com

Содержание

1.	Введение	3
2.	Что такое государственная или частная сеть в схеме блок-цепочки?	5
3.	Что такое блочное программирование?	5
4.	Что такое Термукс?	5
5.	Что такое Мини Блокли Цепь?	6
6.	Архитектура процесса в мини-блочной цепи	10
7.	Схема функциональности BlocklyChain (Mini BlocklyChain).....	14
8.	Что такое Мини БлоклиКод?	15
9.	Установка мини-блокировочной сети связи	16
10.	Синхронизация в узлах системы (мобильный телефон) минуты и секунды.....	38
11.	Конфигурация хранения в Termux	46
12.	Установка сети "Tor" и "Syncthing".....	48
13.	Установка базы данных "Redis" и сервера SSH (Secure Shell).	49
14.	Настройка сервера SSH на мобильном телефоне (смартфоне).....	50
15.	Сетевая конфигурация "Tor" со службой SSH (Secure Shell).....	57
16.	Пиринговая конфигурация системы с ручной синхронизацией.	61
17.	Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).	73
18.	Что такое Доказательство Кванта (PQu)?.....	74
19.	Определение и использование блоков в Mini BlocklyChain	78
20.	Использование блоков для базы данных SQLite (версия MiniSQLite)	105
21.	Определение и использование блоков безопасности.....	109
22.	Установка параметров безопасности в Mini BlocklyChain.....	120
23.	Приложение "Создание баз данных KeyStore & PublicKeys"	124
24.	Приложение "RESTful SQLite GET/POST команды".....	137
25.	Приложение "Java Code SQLite-Redis Connector"	143
26.	Приложение "Mini BlocklyChain для разработчиков".....	146
27.	Приложение "Интеллектуальные контракты с блочным кодом"	158
28.	Приложение "OpenQbit Quantum Computing"	164
29.	Приложение "Расширенные блоки для базы данных SQLite".....	168
30.	Приложение "Пример создания системы Mini BlocklyChain".....	169

31.	Приложение "Интеграция со средствами Ethereum и Биткойн".....	193
32.	Лицензирование и использование программного обеспечения.	213

1. Введение.

Блокчейн обычно ассоциируется с Bitcoin и другими крипто-валютами, но это лишь верхушка айсберга, так как он используется не только для цифровых денег, но и может быть использован для любой информации, которая может иметь ценность для пользователей и/или компаний. Эта технология, которая берет своё начало в 1991 году, когда Стюарт Хабер и В. Скотт Сторнетта описали первую работу над цепочкой криптографически защищённых блоков, не была замечена до 2008 года, когда она стала популярной с приходом bitcoin. Но в настоящее время его использование востребовано в других коммерческих приложениях и, по прогнозам, в среднесрочной перспективе будет расти на нескольких рынках, таких как финансовые учреждения или Интернет вещей IoT среди других секторов.

Блок-схема, более известная как блок-схема, представляет собой единую, согласованную запись, распределенную по нескольким узлам (электронным устройствам, таким как ПК, смартфоны, планшеты и т.д.) в сети. В случае с крипто-валютами, мы можем думать об этом как о бухгалтерской книге, в которой регистрируется каждая из транзакций.

Его работа может быть сложной для понимания, если вникнуть во внутренние детали его реализации, но основная идея проста для понимания.

Он хранится в каждом блоке:

- 1.- количество действительных записей или операций,
- 2.- информация, касающаяся этого блока,
- 3.- его связь с предыдущим и следующим блоком через хэш каждого блока – уникальный код, который был бы похож на отпечаток пальца блока.

Поэтому **каждый блок** имеет **определенное и неподвижное место в цепи**, так как каждый блок содержит информацию из хэша предыдущего блока. Вся цепочка хранится на каждом узле сети, составляющем блок-цепочку, поэтому **точная копия цепочки хранится у всех участников сети**.

По мере создания новых записей они сначала проверяются и подтверждаются сетевыми узлами, а затем добавляются в новый блок, связанный с цепочкой.

Теперь представьте, что эта сеть устройств, которые общаются между собой, имеет возможность взаимодействовать без вмешательства человека, т.е. большое преимущество блок-цепочки в том, что она способна принимать автономные решения, что выгодно в ответ на время обслуживания пользователей, доступное 24 часа в сутки,

минимизирует издержки в бизнесе и в первую очередь имеет уровень безопасности уже проверенный, по этой и другим причинам ставший столь популярным в использовании в различных государственных и частных секторах.

2. Что такое государственная или частная сеть в схеме блок-цепочки?

Общественная сеть. - Это сеть компьютеров или мобильных устройств, которые взаимодействуют друг с другом и поддерживают анонимность, неизвестно, кто формально взаимодействует в этой сети в своих сделках, в этом типе сети любое лицо или компания может взаимодействовать и подключаться в любое время, потому что вам не нужны разрешения на подключение, примером является блок-цепочка Bitcoin, любой может войти, чтобы купить или продать. Обычно этот тип сетей ориентирован или направлен на покупку и продажу цифровых денежных активов или их синонимов - "криптомонии", примеры: DogCoin, Ethereum, LiteCoin, BitCoin, Waves и др.

Частная сеть. - Это сеть компьютеров или мобильных устройств, которые взаимодействуют друг с другом. Однако, в отличие от сетей общего пользования, частные сети требуют предварительного разрешения какой-либо организации (компании или лица), чтобы подключиться и быть частью этого типа сети. Обычно частные блокирующие сети используются в компаниях или корпорациях для осуществления сделок или операций с различными видами информации, которая может иметь ощутимую ценность в виде документов, процессов, разрешений и/или деловых решений, применяемых и контролируемых блоком, примеры: финансовый сектор, страховой сектор, правительство и др.

3. Что такое блочное программирование?

Blockly - это **визуальный язык программирования**, состоящий из простого набора команд, которые мы можем комбинировать, как кусочки головоломки. Это очень полезный инструмент для тех, кто хочет **научиться программировать** интуитивно понятным и простым способом, или для тех, кто уже умеет программировать и хочет увидеть потенциал этого вида программирования.

Блокли - это форма программирования, где не требуется никакого фона ни на каком компьютерном языке, это потому, что это просто соединение графических блоков, как если бы мы играли в лего или головоломку, вам просто нужна какая-то логика и все!

Любой человек может создавать программы для мобильных телефонов (смартфонов), не связываясь с теми языками программирования, которые трудно понять, просто сложите блоки в графическом виде простым, легким и быстрым способом.

4. Что такое Термукс?

Termux - это эмулятор терминалов Android и приложение среды Linux, которое работает напрямую, без необходимости маршрутизации или настройки. Минимальная базовая система устанавливается автоматически.

Мы будем использовать Termux для его стабильности и простоты установки и управления, однако, вы можете использовать установленную среду Ubuntu Linux для Android.

В этой Linux среде у Вас будет "ядро" коммуникационных процессов MiniBlocklyChain.

5. Что такое Мини Блокли Цепь?

Mini BlocklyChain - это полнофункциональный блок-цепь - технология, разработанная для мобильных телефонов с ОС (операционной системой) **Android**, которая будет являться узлами, которые будут выполнять операции по отправке и приему транзакций. Мы создали первую технологию блок-цепочки, которая структурирована "модульным" способом через Blockly программирование, где любой человек с минимальными знаниями и без знания того, как программировать, может создавать и разрабатывать программы для мобильных телефонов и создавать свою собственную блок-цепочку, как в режиме публичной, так и в режиме частной сети. Если вы хотите создать свою собственную цифровую валюту, вы можете сделать это или решение для использования в компании, возможности основаны на потребностях каждого реального случая и бизнес-логики, которую пользователь принимает или создает в соответствии со своими требованиями.

Mini BlocklyChain - это первый модульный блок-цепочка для мобильных телефонов, в которой транзакционная система может быть реализована за короткое время.

Перед определением и использованием "модульных" блоков нам необходимо иметь базовые понятия компонентов Mini BlocklyChain, чтобы знать, когда, как и где их применять в соответствии с реальным случаем, который мы хотим реализовать.

Следующие концепции подчеркивают, на какого типа пользователей они нацелены, поэтому для людей, не обладающих навыками программирования, не важно, чтобы концепции для пользователей, занимающихся разработкой, были полностью понятны.

Основные концепции:

Узел. - Каждое мобильное устройство (телефон, планшет и т.д.) с операционной системой Android, являющееся частью общественной или частной сети Mini BlocklyChain, называется узлом этой сети.

Хэш. - Речь идет о цифровой подписи, которая связана с набором данных, строкой знаков, документами или какой-либо цифровой информацией, эта цифровая подпись уникальна и неповторяется, что помогает отправлять или получать информацию, чтобы изначальное содержание отправленной информации не могло быть изменено.

Активный. - Любые виды цифровых данных или информации, которые могут быть взвешены с некоторой материальной или нематериальной ценностью для людей или компаний (документы, цифровые монеты, музыка, видео, изображения, цифровые разрешения и т.д.).

Сделка. - Обмен информацией между узлами, занимающими тот или иной актив.

UXTO Transaction - это тип транзакции, который упаковывает одну или несколько транзакций из узлов, и все неизрасходованные транзакции из каждого узла (UXTO: Выходы неизрасходованных транзакций) могут быть потрачены как входные данные в новой транзакции. Эти транзакции группируются в очередь, которая обрабатывается каждый определенный момент времени, что может регулироваться в соответствии с требованиями каждого информационного потока.

Операция (ввод). - Это вид операции, основанный на операциях "UXTO". Когда очередь операций "UXTO" поступает, она обрабатывается **входной операцией**, которая классифицирует операцию как депозит или расход и, таким образом, может иметь сальдо конечного результата для каждого пользователя.

Адрес источника. - Это адрес человека, который генерирует или отправляет транзакцию для обработки в очереди SQLite Master. Это алфавитно-цифровое число из 64 символов, которое создается и верифицируется системой.

Адрес назначения. - Это адрес лица, которое получает транзакцию и добавляет ее на свой баланс или хранит отправленный актив. Это 64-символьный буквенно-цифровой номер, который создается и верифицируется системой.

SQLite Master. - API REST база данных, которая получает транзакции и создает очередь для обработки каждой определенной переменной или фиксированного времени в соответствии с потребностями бизнеса.

Транзакция по базе данных. - Это транзакции, которые отражаются в базе данных SQLite, которая резервирует или хранит информацию о транзакциях Mini BlocklyChain.

AES (Advanced Encryption Standard) - Это процесс безопасности, который шифрует данные, хранящиеся в базе данных SQLite Mini BlocklyChain.

Баланс. - После осуществления любого процесса транзакции в Мини-БлоклыЦене, баланс операции получается либо как материальная (криптомонета), либо как нематериальная ценность (бизнес-процессы между людьми или компаниями).

Бизнес-процесс. - Речь идет о любом процессе, связанном с каким-либо потоком информации для получения результата конечным пользователем, конечным пользователем может быть лицо или компания из различных секторов частного или государственного сектора.

Государственный и частный ключ. - Это вид шифрования информации, при котором два буквенно-цифровых ключа, связанных друг с другом, генерируются и используются для отправки конфиденциальной информации через государственные или частные сети. Частный ключ используется для шифрования информации, и при отправке по сети он не может быть изменен или разборчив с первого взгляда, публичный ключ - это тот, который совместно используется и который виден любому лицу или компании, и этот ключ поможет проверить отправленную информацию, а также сможет прочитать ее только действительному адресату.

Цифровая подпись. - Это подпись, которая может быть создана с помощью частного ключа, при помощи этой подписи получатель гарантирует, что информация не была изменена, и подтверждает, что информация действительна для получателя.

Цифровой адрес (адрес мини-блочковой цепи). - Это адрес, который состоит из буквенно-цифровых символов, уникальных для каждого пользователя Mini BlocklyChain, этот адрес используется для выполнения транзакций между пользователями и независимо от того, государственная это или частная сеть.

Волшебный номер. - Это случайное число, определяемое бизнес-правилами для каждой выполняющейся UXTO-транзакции, которое служит для авторизации узла в качестве кандидата на выполнение UXTO-транзакции и создания нового блока Mini BlocklyChain.

Создание нового блока. - Новый блок в Mini BlocklyChain представляет собой хэш, созданный и прикрепленный узлом, который вышел в качестве выигравшего кандидата на обработку транзакции UXTO.

Протокол консенсуса PoW (доказательство работы). - Это тест, который выполняет все узлы, и первым узлом, который успешно завершает тест, является тот, который выбран для выполнения транзакции UXTO. Это алгоритм, который состоит из математических вычислений для получения результата (хэша) по правилам, заданным в каждой транзакции, выполняемой Mini BlocklyChain, основывается на износе или

востребованности ресурсов обработки информации компьютеров, в случае, если Mini BlocklyChain был структурирован и модифицирован для получения "магического числа" это число, чтобы иметь авторизацию или консенсус большинства узлов, чтобы быть тем, кто может выполнить транзакцию UXTO. У PoW максимальный уровень сложности 5, так как он используется только для получения "магического числа".

Протокол консенсуса PoQu (квантовый тест) - Это тест, который запускает все узлы и который изначально составляется PoW для получения "магического числа", а затем выполняет алгоритм вероятности на основе QRNG (Quantum Random Number Generator - Генератор квантовых случайных чисел) - генератора квантовых случайных чисел. Этот процесс обеспечивает равенство вероятности для всех узлов и поэтому выбирает, какой узел будет выполнять операцию UXTO и создание нового блока.

Дерево Меркл. - Это метод безопасности, чтобы гарантировать Mini BlocklyChain, что транзакции являются действительными или не были изменены любой внешней организацией. Дерево хэш-меркл - это бинарная или небинарная структура дерева данных, в которой каждый узел, не являющийся листом, помечен хэшем конкатенации меток или значений его дочерних узлов. Они являются обобщением хэш-списков и хэш-строк.

Redis DB. - База данных транзакций в режиме реального времени, используемая для обработки и отправки на узлы новых запрашиваемых транзакций.

SQLite DB. - База данных, где хранятся балансы и новые блоки для Mini BlocklyChain для обеспечения целостности сети.

Часовой. - Коннектор безопасности и целостности данных между Redis и SQLite. Этот разъем или программа отвечает за просмотр, обработку, валидацию, распределение и перевод транзакций на узлы.

OpenSSH. - Разъем безопасности для выполнения задач внутри операционной системы Android.

Термекс Шелл Терминал. - Программа, в которой вы можете найти зависимости третьих лиц для обработки транзакций Mini BlocklyChain, в этой версии 1.0 она используется для легкой установки, однако в будущих версиях она будет заменена системой BlocklyShell, которая в настоящее время находится в разработке.

Бумажник. - Именно в цифровом хранилище хранятся два фундаментальных данных по любой сделке; публичный и частный ключи, которые будут использоваться в качестве адреса источника и цифровой подписи, соответственно, по любой совершенной сделке, а также баланс отправленных или полученных транзакций. Это хранилище, в котором хранятся адреса Мини-БлоклиЦенейна, а также результаты операций UXTO (Баланс).

Разработчик приложений или программист:

Объектно-ориентированное программирование (Java) - Мини BlocklyChain сделан на Java.

Блокли코드. - Это термин интеллектуальных контрактов, которые могут быть выполнены в среде Mini BlocklyChain, BlocklyCode создается в java-программировании.

OpenJDK для Android. - Это набор JDK и JRE для Android, где создается и выполняется BlocklyCode.

QRNG (Генератор квантовых случайных чисел). - Это генератор квантовых случайных чисел, Mini BlocklyChain в настоящее время имеет два API для их генерации.

Rsync. - Это бесплатное приложение для систем типа Unix и Microsoft Windows, которое обеспечивает эффективную передачу инкрементных данных, а также работает со сжатыми и зашифрованными данными.

ffsend. - Это приложение для простого и безопасного обмена файлами из командной строки.

НТП. - Протокол сетевого времени, представляет собой интернет-протокол для синхронизации часов компьютерных систем через пакетную маршрутизацию в сетях с переменной задержкой.

Термукс (развитие). - Терминал Shell с широким спектром приложений и библиотек с открытым исходным кодом.

Блочные модули (данные). - Разработчики могут интегрировать модули для расширения возможностей Mini BlocklyChain.

Пирс - Пирс. - Этот термин относится к взаимодействию между узлами в прямом смысле, то есть обновление информации не зависит от центрального сервера, но каждый узел работает как центральный сервер, который взаимодействует между всеми узлами, имея одну и ту же информацию, что помогает избежать точек отказа.

Синхронизация. - инструмент для синхронизации данных или файлов между двумя устройствами с использованием типа связи "Peer to Peer".

Красный Тор. - Это сеть распределенных коммуникаций с низкой задержкой, накладываемая в Интернете, в которой маршрутизация сообщений, которыми обмениваются пользователи, не раскрывает их личности, т.е. их IP-адреса (сетевая анонимность).

Мобильная маршрутизация. - Процесс установки внешнего программного обеспечения на телефон, чтобы войти в качестве системного администратора в операционной

системе Linux (Android) администратор пользователь называется "корень", чтобы иметь возможность вращать телефон будет иметь доступ к любому процессу. Важно отметить, что некоторые производители мобильных телефонов (смартфонов) заявляют, что гарантия теряется из-за любой неисправности мобильного телефона.

6. Архитектура процесса в мини-блочной цепи

Mini BlocklyChain формируется тремя процессами, которые формируют его архитектуру, первый - это бизнес-процессы, второй - коммуникационные процессы, а третий - среда разработки дополнительных модулей и/или создание BlocklyCode "интеллектуальных контрактов".

Бизнес-процессы представляют собой блоки, которые формируют серию процедур для создания транзакции либо в публичной, либо в частной сети, этот тип процесса представляет собой планирование бизнеса, как, когда, что, кто, где и другие атрибуты будут упорядочены, спланированы и распределены таким образом, чтобы была достигнута основная функциональность каждой отправляемой, получаемой, хранимой и/или отвергаемой транзакции. Первый процесс состоит из следующих типов блоков, разделенных на четыре категории:

1. Блоки ввода данных
2. Блоки обработки данных
3. Блоки безопасности.
4. Модульные блоки данных (разработчики)
5. Боты связи.

Блоки ввода данных - это те блоки, которые получают транзакцию как минимум с четырьмя входными параметрами (**адрес источника, адрес назначения, активный, данные**) и могут иметь больше в зависимости от переменной "данные", это может быть блок, разработанный третьими лицами, или с нулевым значением (NULL).

Блоки обработки данных разрабатывают логистику, расчеты, нормализацию данных, логические решения и проверку потоков для каждой транзакции. Эти типы блоков используются для придания интеллекта бизнес-процессу и основаны, главным образом, на его названии, на обработке всех типов информации, а также на ее преобразовании из различных типов данных.

Блоки безопасности используются для проверки информации и обеспечения того, чтобы транзакции не были изменены от их источника до места назначения, они всегда обрабатываются с помощью различных алгоритмов безопасности, используемых в технологиях блок-цепочки, среди наиболее используемых инструментов являются (в

частности, хэш-подпись, цифровая подпись, шифрование данных AES, создание и проверка достоверности цифровых адресов).

Модульные блоки данных, это блоки, которые разрабатываются третьими лицами, помните, что Mini BlocklyChain был создан по модульному принципу, чтобы быть обогащенным новыми блоками в соответствии с потребностями каждого сектора, будь то государственный или частный. Чтобы узнать больше о том, как создавать модули, ознакомьтесь с Приложением Разработчики.

Как мы уже отмечали ранее, архитектура Mini BlocklyChain во втором компоненте представляет собой процессы коммуникаций, именно эти процессы отвечают за каналы связи по TCP и Sockets связи, чтобы обеспечить гибкость при отправке и получении транзакций либо различными средствами, которые используются в данный момент наиболее часто: Мобильный Интернет, WiFi в настоящее время работает для включения средства связи Bluetooth.

Блоки связи, в основном, основаны на обмене данными с безопасностью, реализованной в канале передачи, и основаны на связи по протоколу SSH (Secure Shell) с различными этапами, через которые проходит отправленная или полученная транзакция.

Коммуникационная часть является фундаментальной, поскольку эти процессы выполняют функцию обновления информации во всех узлах по TCP, а соединение, называемое "**Peer to Peer**", блоки, которые вмешиваются в коммуникацию, основаны на обмене информацией между узлами без вмешательства серверов-посредников, так что каждый узел делает его независимым, имея возможность создать сеть узлов, где точки отказа минимальны или почти нулевые. Точно так же такая же независимость каждого узла помогает им принимать решения индивидуально или в целом в соответствии с потребностями бизнеса.

Архитектура "Peer to Peer" состоит из трех частей, образующих общедоступную или частную сеть, которую вы решили создать, в обоих случаях канал связи шифруется от узла к узлу.

Первым компонентом для связи между мобильными устройствами (смартфонами) или wifi является обеспечение узлов или устройств сетью, в которой они могут быть найдены в любой точке мира, сеть связи, в которой установлен MiniBlocklyChain - это сеть "**Tor**".

Что такое сеть "Top"? - (<https://www.torproject.org>)

"**Tor** (аббревиатура от Te Onion Router - на испанском языке) - проект, основной целью которого является создание распределенной коммуникационной сети с низкой задержкой, наложенной на Интернет, в которой маршрутизация сообщений, которыми обмениваются

пользователи, не раскрывает их личности, т.е. их IP-адреса (анонимность на сетевом уровне), и которая, кроме того, сохраняет целостность и конфиденциальность информации, проходящей через нее".

Вторая составляющая и не менее важная задача - добиться того, чтобы все узлы в MiniBlocklyChain в любое время имели одинаковые данные или чтобы их базы данных и файлы синхронизировались для выполнения этой задачи между узлами будет реализована **"синхронизация"**.

Что такое сеть Syncing? - (<https://syncing.net>)

Syncthing - бесплатное приложение с открытым исходным кодом для синхронизации одноранговых файлов, доступное для Windows, Mac, Linux, Android, Solaris, Darwin и BSD. Вы можете синхронизировать файлы между устройствами в локальной сети или между удаленными устройствами через Интернет.

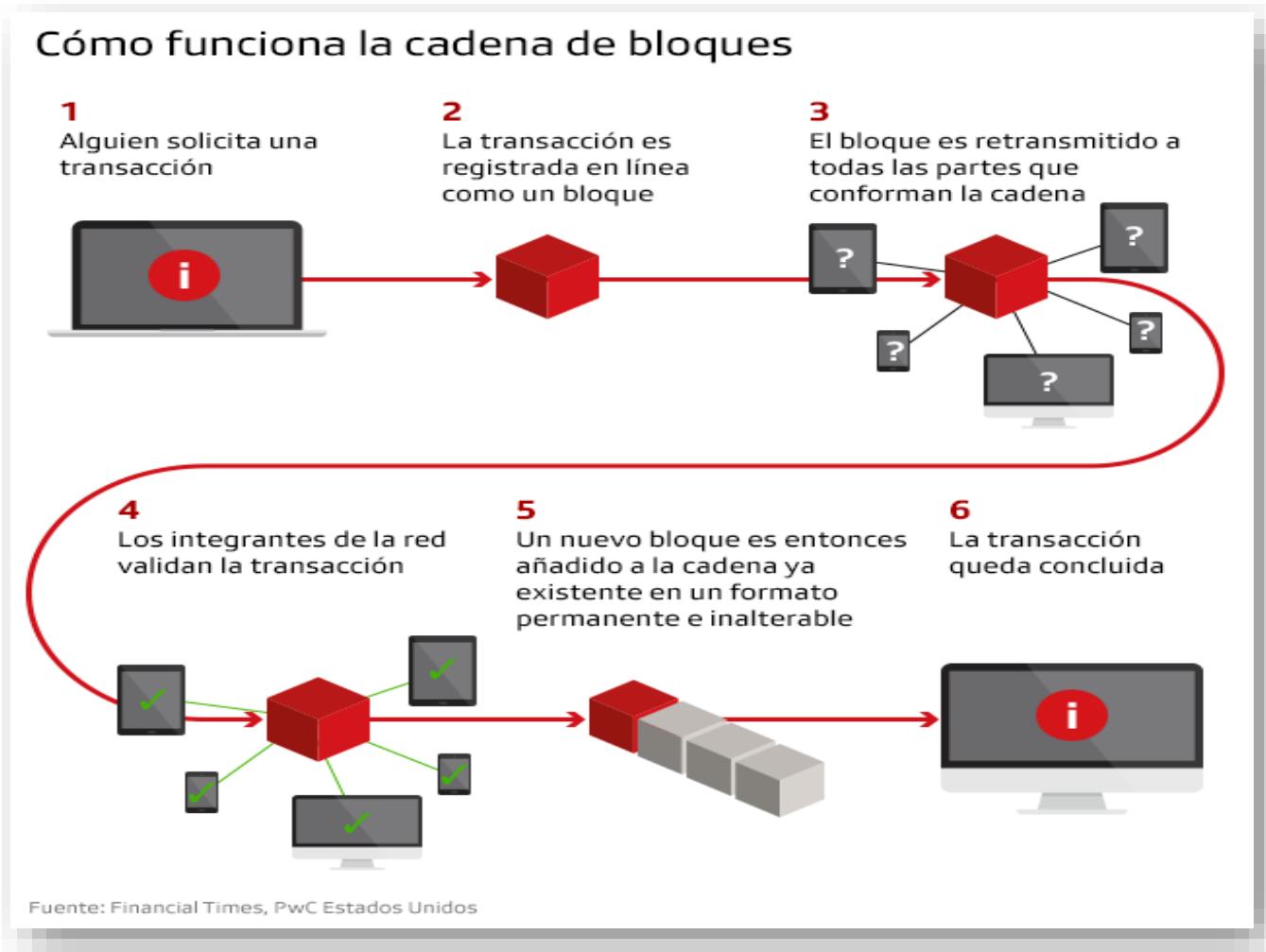
На основе двух предыдущих компонентов связи мы можем начать разработку сети доверия между узлами и с уровнем синхронизации данных, который будет основой или "ядром" бизнес-процессов для удовлетворения любых потребностей пользователя или компании.

Третьим компонентом в сети связи является база данных Redis, которая будет уведомлять все узлы в режиме реального времени о новых поступающих и отправляемых транзакциях.

Что такое сеть Redis DB? - (<https://redis.io>)

Redis - это движок базы данных в памяти, основанный на хранении в хэш-таблицах (ключ/значение), но который по желанию может быть использован в качестве долговечной или постоянной базы данных.

7. Схема функциональности BlocklyChain (Mini BlocklyChain).



8. Что такое Мини БлоклиКод?

Mini BlocklyCode - это программы, созданные на языке Java и хранящиеся в репозитории независимо от узлов, они обычно называются интеллектуальными контрактами, эти программы сконструированы с логическими условиями так, что при выполнении этих условий они выполняются автоматически, без зависимости от каких-либо авторизаций или внешнего вмешательства человека.

"Умный" контракт - это компьютерная программа, которая облегчает, обеспечивает безопасность, обеспечивает исполнение и выполняет зарегистрированные соглашения между двумя или более сторонами (например, физическими лицами или организациями). Как таковые, они будут помогать им в переговорах и определении таких соглашений, которые приведут к определенным действиям в результате выполнения ряда конкретных условий.

Умный контракт - это программа, которая живет в системе, не контролируемой ни одной из сторон, ни их агентами, и которая выполняет автоматический контракт, работающий как если бы это было предложение любой другой компьютерной программы. С той разницей, что это делается таким образом, чтобы взаимодействовать с реальными активами. Когда срабатывает заранее запрограммированное условие, не подлежащее никакому человеческому суждению, интеллектуальный контракт исполняет соответствующее положение контракта.

Они направлены на обеспечение большей безопасности, чем традиционное договорное право, и снижение операционных издержек, связанных с заключением контрактов. Передача цифровых ценностей через систему, не требующую доверия (например, bitcoins), открывает путь к новым приложениям, которые могут использовать интеллектуальные контракты. "

Mini BlocklyCode предназначен для разработчиков с опытом программирования на Java. В Mini BlocklyChain некоторые типы библиотек ограничены в целях безопасности одной и той же системы, однако могут быть созданы библиотеки для создания транзакций по отправке или получению некоторой договорной стоимости, созданной или согласованной двумя или более сторонами.

Каждая мини-блочная цепь соответствует следующим требованиям:

- ✓ Любой созданный Mini BlocklyChain основан на запущенных сообщениях, а не на исполнениях в системе, однако, эти сообщения могут содержать авторизации, физические одобрения договоров или физические действия.
- ✓ Каждый созданный Mini BlocklyChain должен проходить через коммуникационный блок "аудитор", который отвечает за мониторинг вредоносного или запрещенного кода для просмотра предложений или несанкционированных заказов в системе.
- ✓ Все Mini BlocklyChain выполняется только на JVM исходного узла, программы не выполняются глобально для защиты системы.

Подробнее см. приложение "Mini BlocklyChain для разработчиков".

9. Установка мини-блокировочной сети связи

1. Установка и настройка сети Mini SQLSync

Сеть Mini SQLSync отвечает за получение транзакций от узлов, чтобы они могли обрабатывать эти транзакции. Эта сеть формируется основной сетью, которая через "Peer to Peer" между узлами и резервной сетью, называемой Mini Sentinel RESTful.

Мы начнем с установки RESTful Mini Sentinel сети резервного копирования, эта служба является той, которая будет получать транзакции от узлов, эта служба основана на хранении транзакций в течение определенного времени, чтобы впоследствии преобразовать информацию через коннектор, который будет впрыскивать очередь из всех зашифрованных транзакций в службу Redis. Позже служба базы данных Redis выполнит в режиме реального времени выпуск зашифрованной очереди транзакций на все узлы, интегрирующие сеть Mini BlocklyChain.

Услуга или дизайн типа RESTful - это простой и удобный способ посмотреть, изменить, удалить и/или вставить информацию в базу данных через URL или интернет-адрес, в нашем случае мы будем использовать базу данных SQLite, так как она характеризуется тем, что представляет собой всю информацию в одном файле. Для использования требований с необходимостью большого масштабирования мы сможем использовать другой тип БД, как MySQL, Oracle, DB2 или другую коммерческую БД, просто нам придется поменять коннектор Mini BlocklyChain SQLite (бесплатная версия) на коннектор Mini BlocklyChain DB других (коммерческая версия).

ВАЖНОЕ ЗАМЕЧАНИЕ: RESTful Mini Sentinel конфигурация не обрабатывает любые виды транзакций в любое время, это только служба, которая форматирует "формирует информацию", так что узлы могут выполнять обработку транзакций должным образом и в запланированное и синхронизированное время для всех узлов.

Установка сервиса RESTful основывается на базе базы данных SQLite. Эта установка будет производиться в среде Windows для практических целей, однако эта модель может быть легко реплицирована в операционной системе типа Linux.

Для тех, кто хочет просмотреть производительность и поддержку запросов и вставок SQLite, обратитесь к этим тестам производительности:

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

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

Установка сети резервного копирования RESTful Mini Sentinel. Мы будем устанавливать и настраивать эту услугу на компьютере с Windows 10, однако, процесс также был установлен в среде Ubuntu 18.09 Linux легко.

Требования:

- ✓ Сервер баз данных SQLite в режиме RESTful (<https://github.com/olsonpm/sqlite-to-rest>).
- ✓ SQLite-коннектор Sentinel Redis
- ✓ Переписать сервер базы данных в режиме Master-Slave (<https://redis.io/topics/replication>)

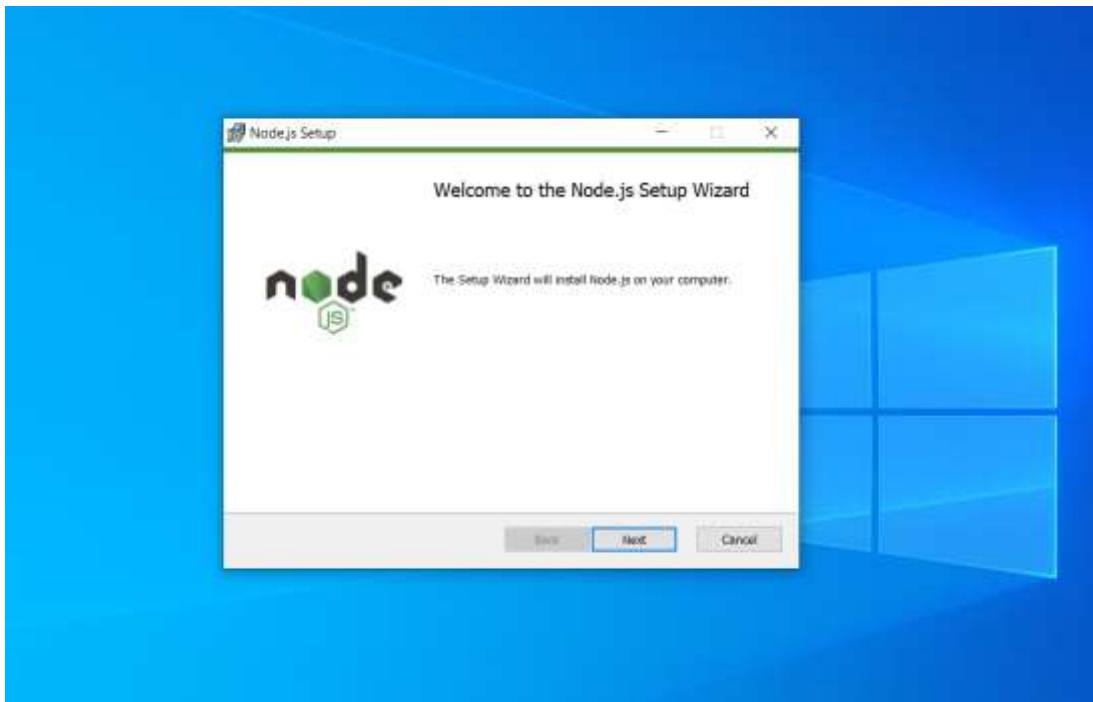
Установка сервера баз данных SQLite в режиме RESTful

Для установки необходимо начать со следующих программных пакетов: Nodejs, sqlite3 и Git Bash для Windows.

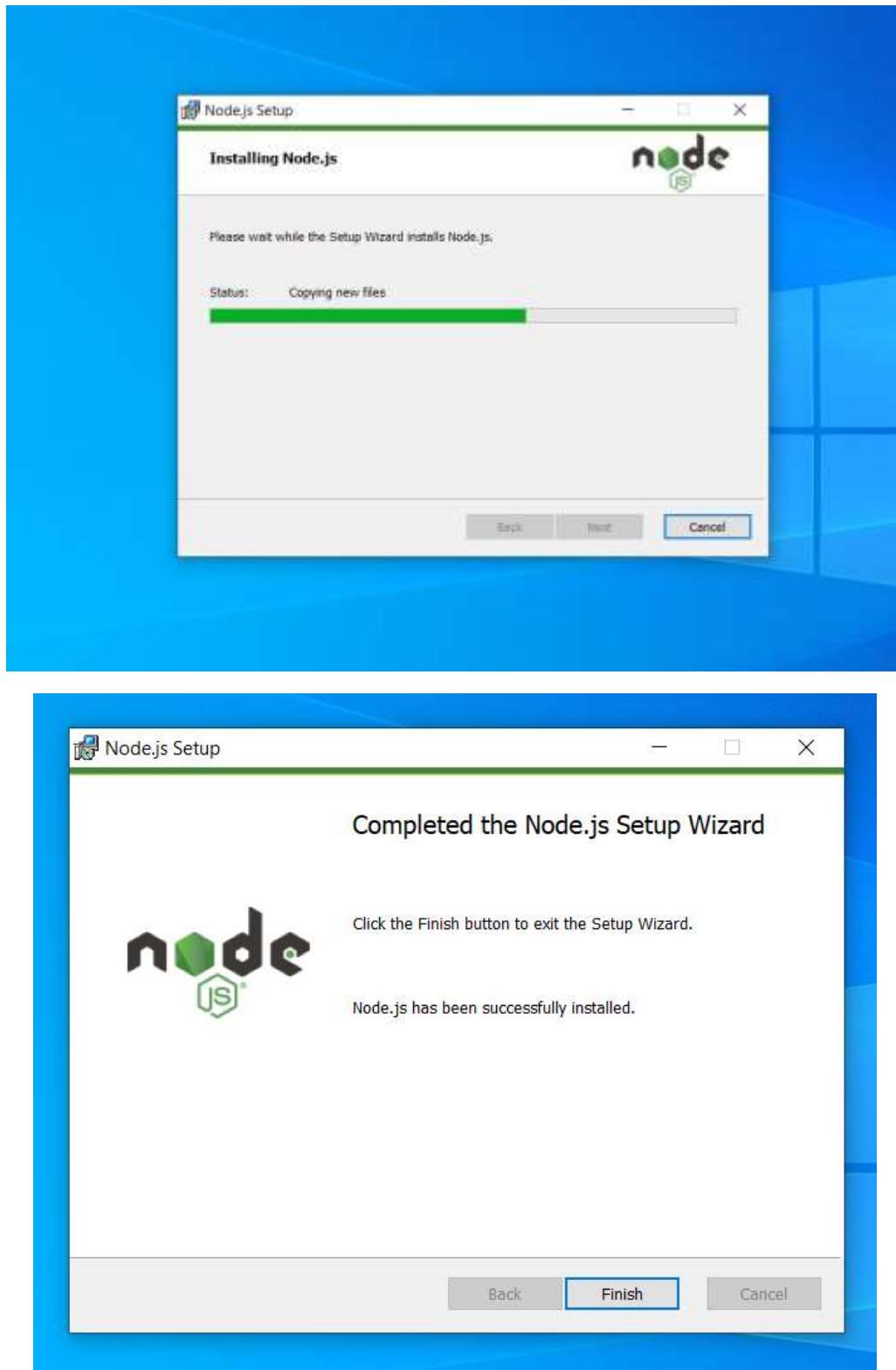
Для получения Nodejs мы посмотрели их официальный сайт <https://nodejs.org/es/> и выбрали версию "Рекомендуется для большинства".



После загрузки файла с расширением .msi мы дважды щелкаем для его установки.



Мы выполняем установку по умолчанию, просто нажимаем "Далее", не выбирая никаких дополнительных опций.



После завершения установки Nodejs мы продолжаем установку базы данных SQLite.

Мы заходим на их официальный сайт <https://www.sqlite.org/index.html> и нажимаем на ту часть, где вы "скачиваете".



What Is SQLite?

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured database engine in the world. SQLite is built into all mobile phones and most computers and is used by millions of people every day. [More Information...](#)

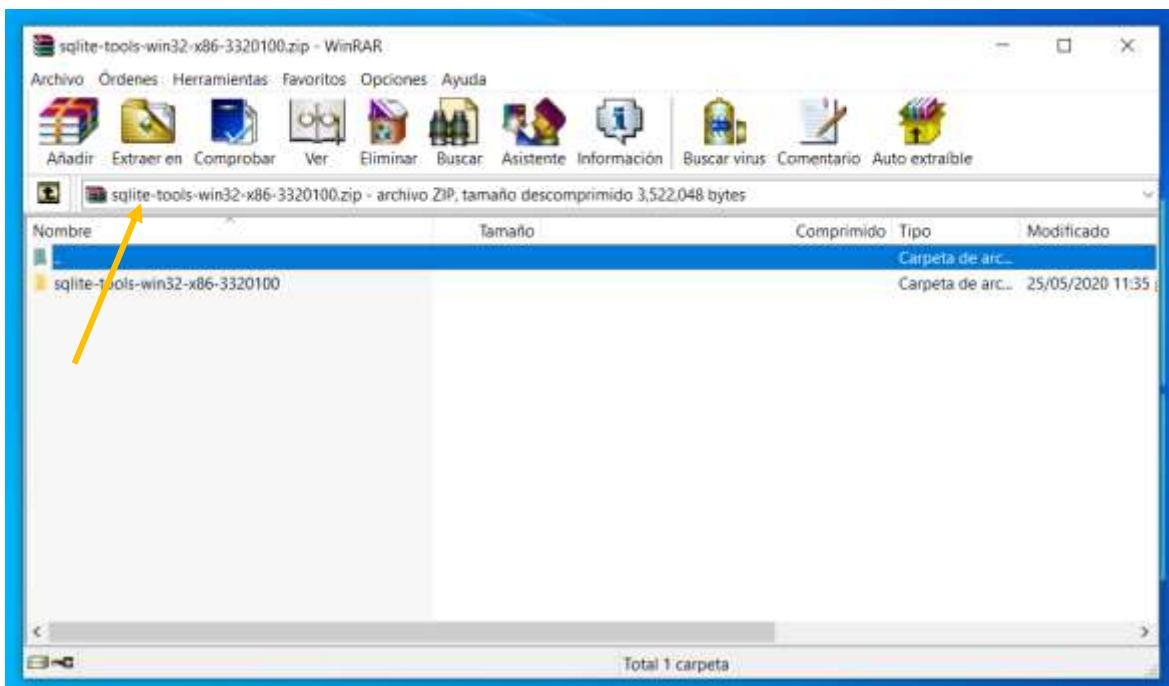
The SQLite [file format](#) is stable, cross-platform, and backwards compatible and the developers pledged to support it until at least 2050. SQLite database files are commonly used as containers to transfer rich content between systems. The file format is designed to be a simple and efficient way to store data [4]. There are over 1 trillion (1e12) SQLite databases in active use [5].

SQLite [source code](#) is in the [public-domain](#) and is free to everyone to use for any purpose.

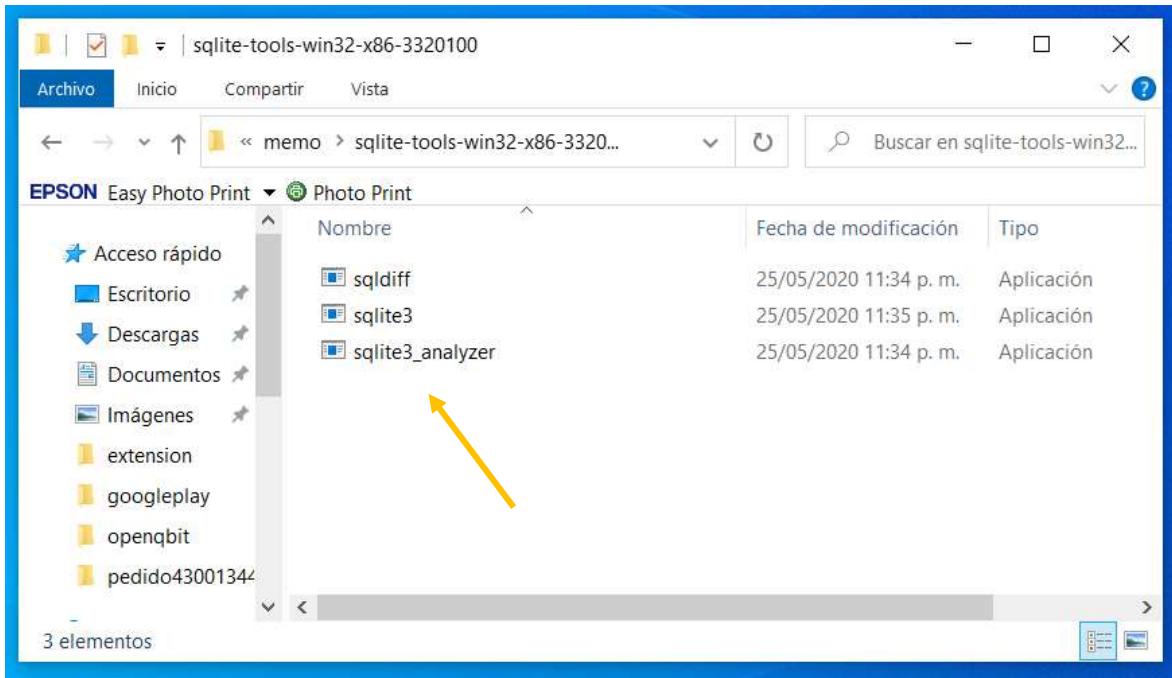
Latest Release

[Version 3.32.1 \(2020-05-25\)](#). [Download](#) [Prior Releases](#)

Далее выбираем опцию, где написано "Precompiled Binaries for Windows" и нажимаем на версию программного обеспечения "sqlite-tools-win32-x86-3320100.zip" по окончании загрузки, переходим к распаковке файла простым способом, открываем папку, в которую был скачан файл и дважды нажимаем на файл для его распаковки.

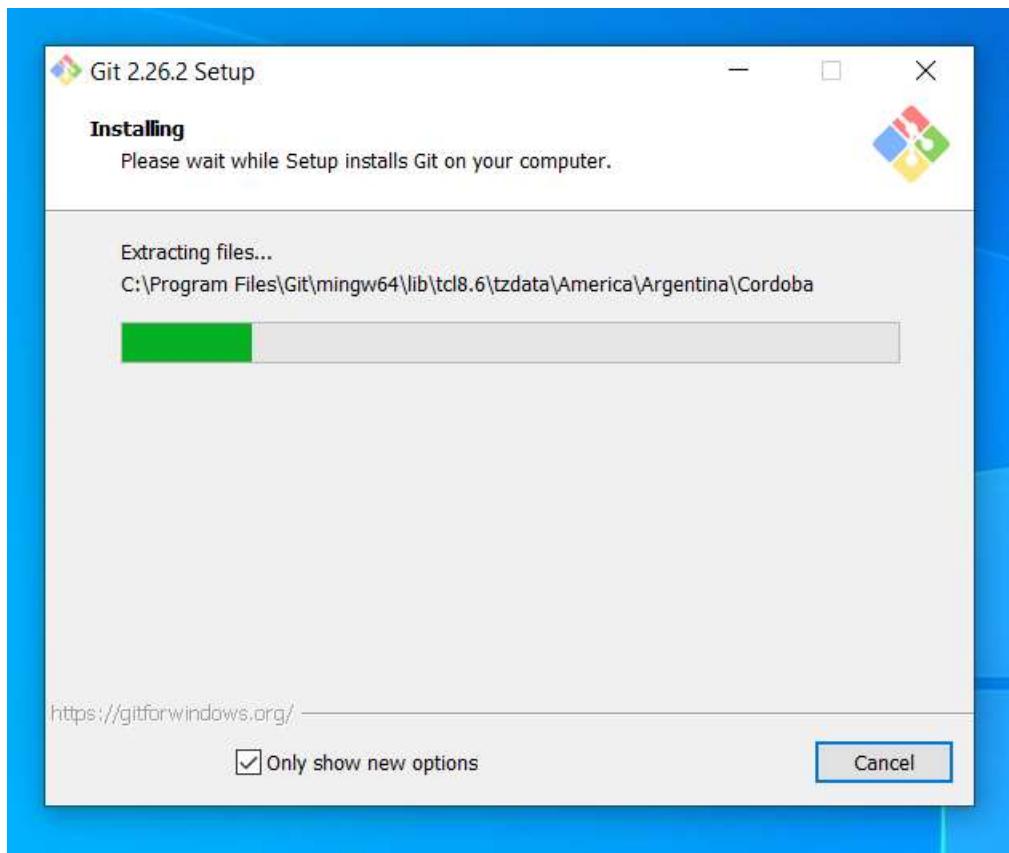


После распаковки у нас будет три файла, это наша среда для создания базы данных SQLite, которую мы будем использовать позже.



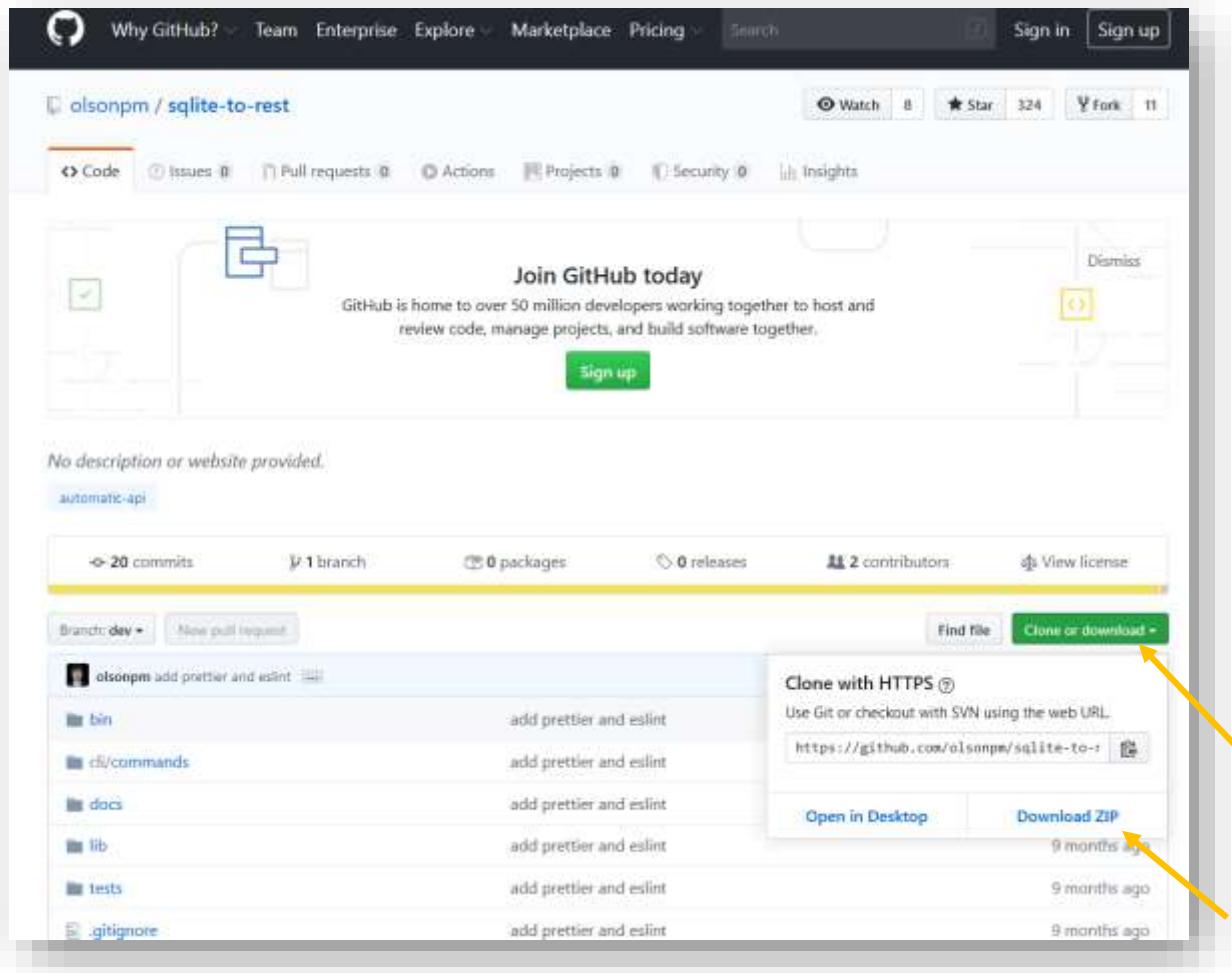
Мы начнем с установки Git Bash, подобного Linux эмулятора терминала, который поможет нам запустить RESTful backup сервис должным образом.

Мы скачаем его с официального сайта <https://gitforwindows.org/> и приступим к его установке с указанными в нем опциями по умолчанию.



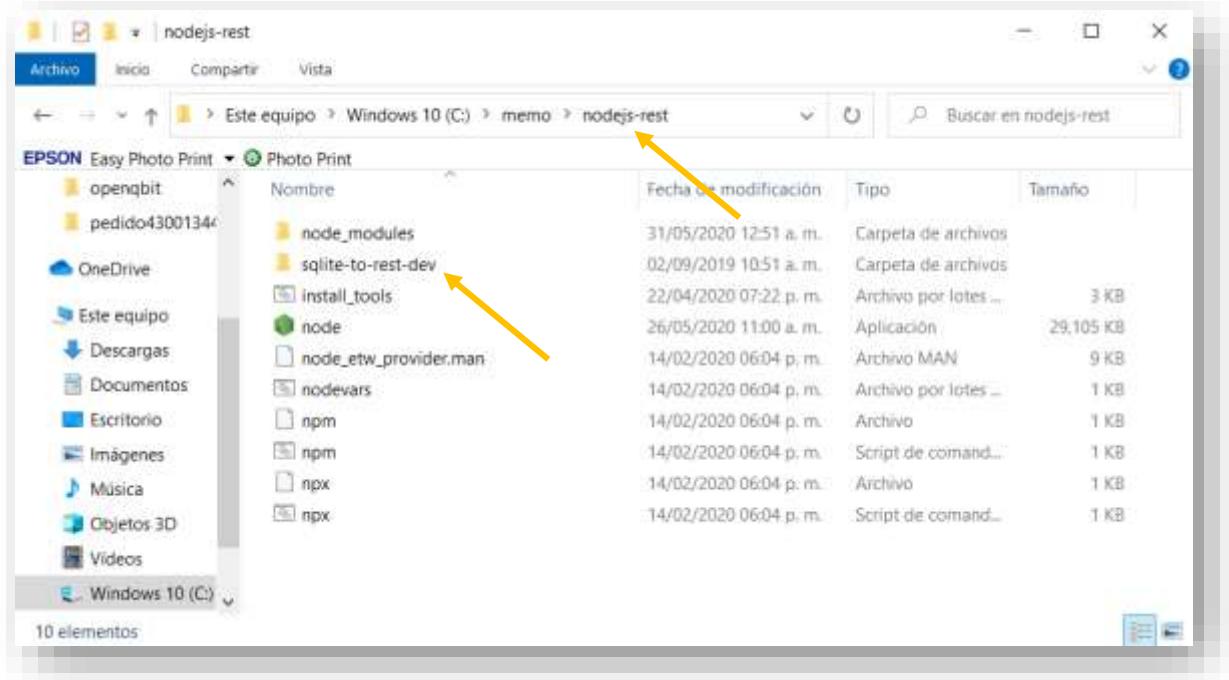
Теперь, когда у нас на машине с Windows 10 установлены nodejs, sqlite и Git Bash, мы приступим к установке среды, которая будет поддерживать нашу базу данных SQLite в режиме RESTful.

На момент загрузки программного обеспечения, которое предоставит функционал RESTful для SQLite. Для этого мы должны перейти на следующий сайт, <https://github.com/olsonpm/sqlite-to-rest>. В нем мы нажмем на зеленую правую кнопку "Клонировать или скачать" и выберем опцию "Загрузить ZIP", которая загрузит программное обеспечение на нашем компьютере.



После скачивания в нашем компьютере мы продолжаем распаковывать его внутри каталога или папки, в которую мы установили программу **nodejs**, и у нас появится каталог с именем "**sqlite-to-rest-dev**".

ЗАМЕЧАНИЕ: Важно, чтобы каталог **sqlite-to-rest-dev** находился внутри каталога, в котором был установлен **nodejs**.

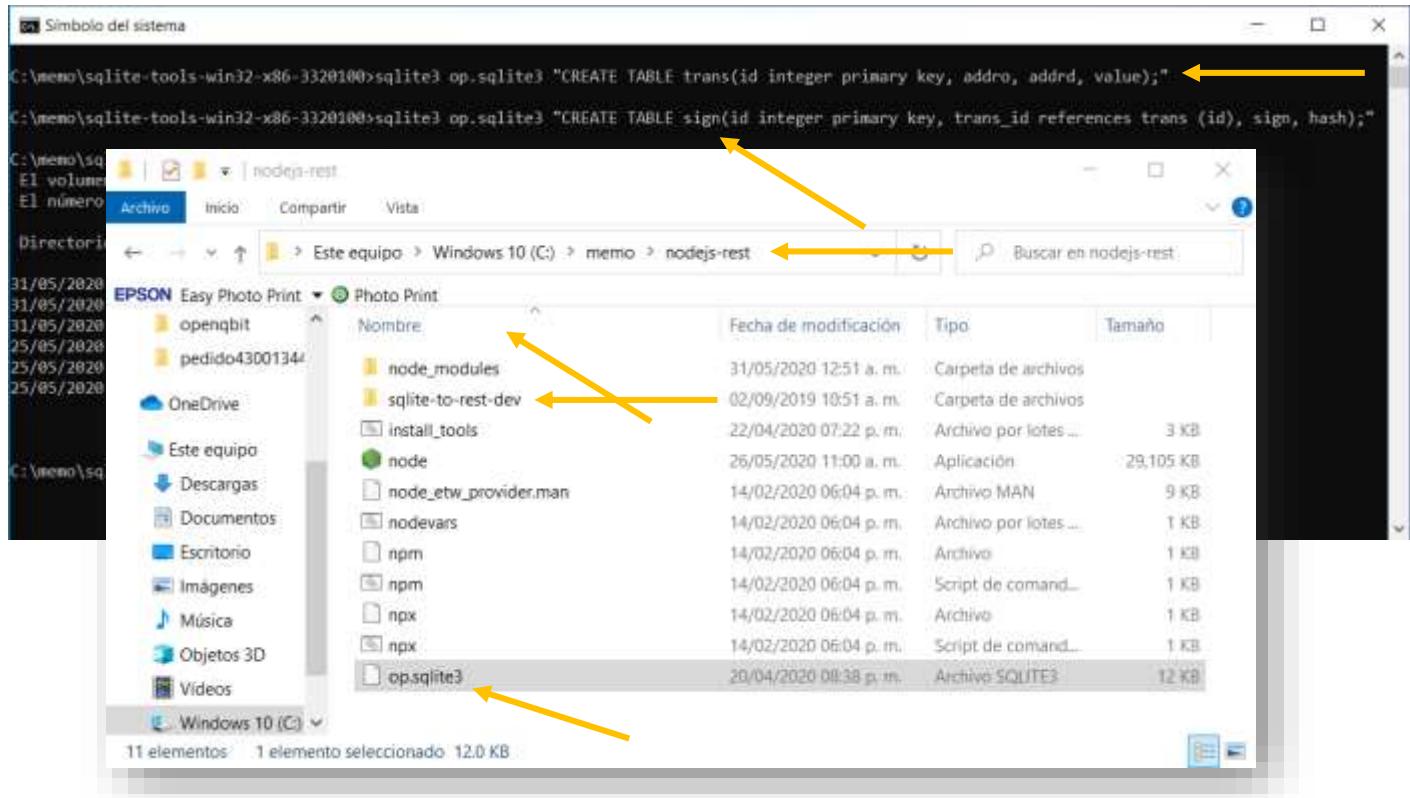


После того, как все установлено, мы приступаем к настройке базы данных SQLite, которую будем использовать для хранения транзакций узлов в среде резервного копирования RESTful.

Дизайн таблиц и структура данных. Команды, которые должны выполняться в режиме онлайн в списке команд CMD

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

sqlite3 op.sqlite3 "Знак CREATE TABLE (id целый первичный ключ, trans_id ссылки trans (id), знак, хэш);".

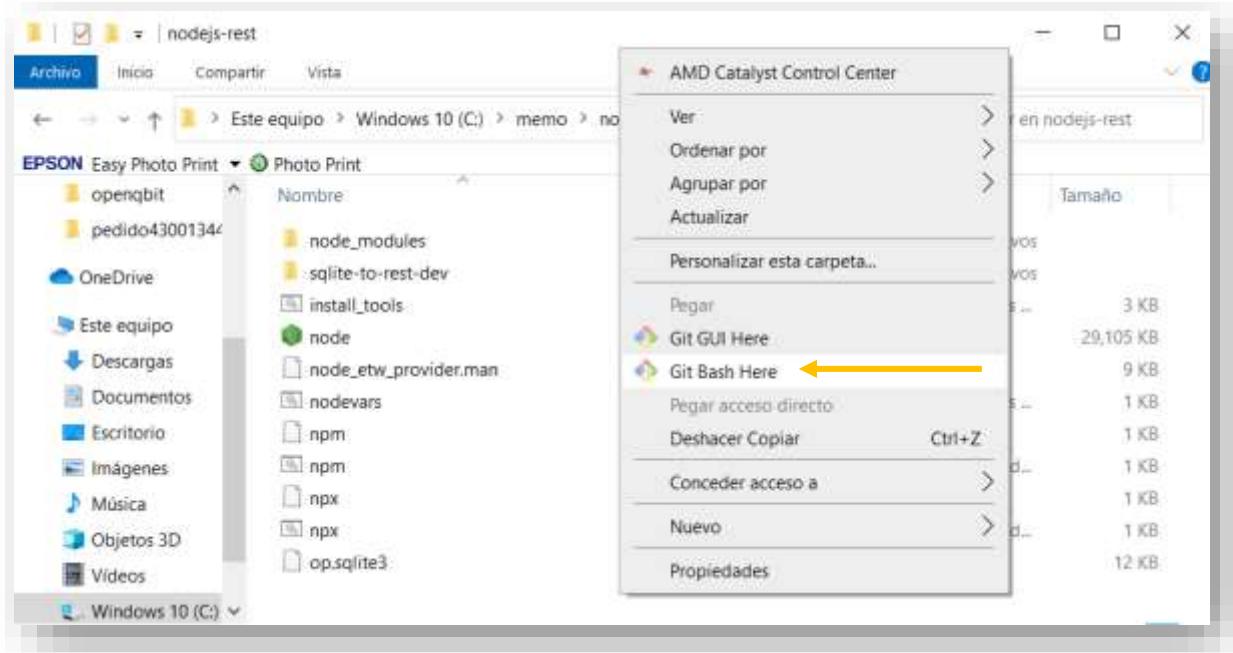


После создания базы данных op.sqlite3 мы должны сделать копию в каталоге, где были установлены **узлы**. В каталоге nodejs также должна быть копия "sqlite-to-rest-dev".

Мы помещаем себя в папку, где находится установка **nodejs** и где также должно быть программное обеспечение "**sqlite-to-rest-dev**". Наведите курсор на папку с указателем и щелкните правой кнопкой мыши, чтобы открыть меню и выберите, где написано "**Git Bash**", чтобы открыть терминал.

В новом открытом терминале Git Bash мы выполняем следующие команды:

```
$ 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 установка --global 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
$
```

После выполнения команды появится установка пакета "sqlite-to-rest".

Мы сгенерировали RESTful среду для SQLite с базой `op.sqlite3`, которую мы создали ранее.

Выполняем команду: \$ sqlite-to-rest генерируем скелет --db-path ./op.sqlite3



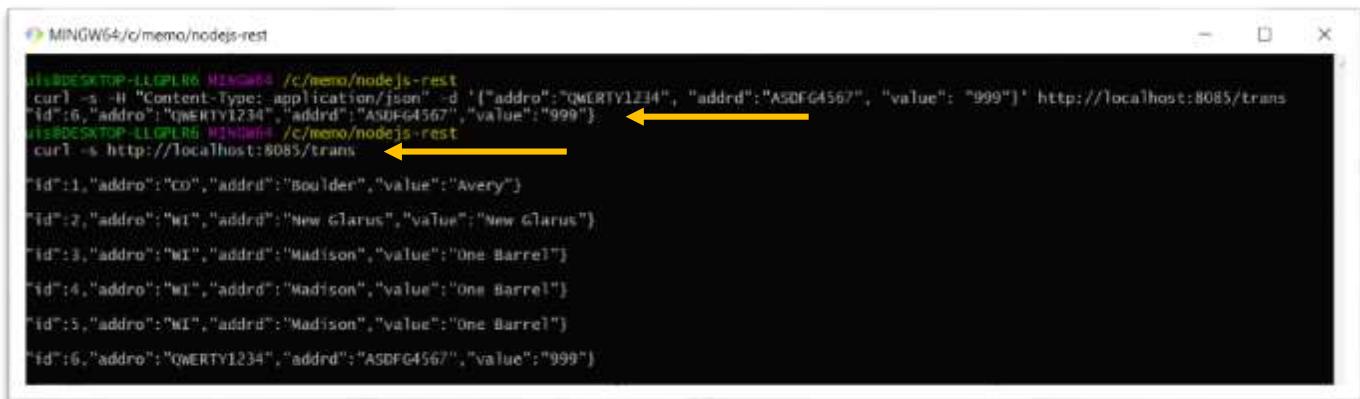
```
MINGW64/c/memo/nodejs-rest
u1s0DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
sqlite-to-rest generate-skeleton --db-path ./op.sqlite3
package.json found in working directory.
Installing dependencies...
Writing the skeleton server to: skeleton.js
finished!
u1s0DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
```

Мы запустили RESTful SQLite сервис на порту 8085 по умолчанию. Мы запускаем следующую команду: \$ узел скелет.js



```
MINGW64/c/memo/nodejs-rest
u1s0DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
node skeleton.js
listening on port: 8085
```

Мы протестируем услугу RESTful с добавлением данных и запросом данных.



```
MINGW64/c/memo/nodejs-rest
u1s0DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
curl -s -H "Content-Type: application/json" -d '{"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}' http://localhost:8085/transactions
u1s0DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
curl -s http://localhost:8085/transactions
```

The terminal output shows the following JSON data added to the database:

```
{"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"}
```

Для тестирования сервиса SQLite RESTful мы используем следующие команды:

Вставить данные в таблицу trans, находящуюся в БД op.sqlite3:

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

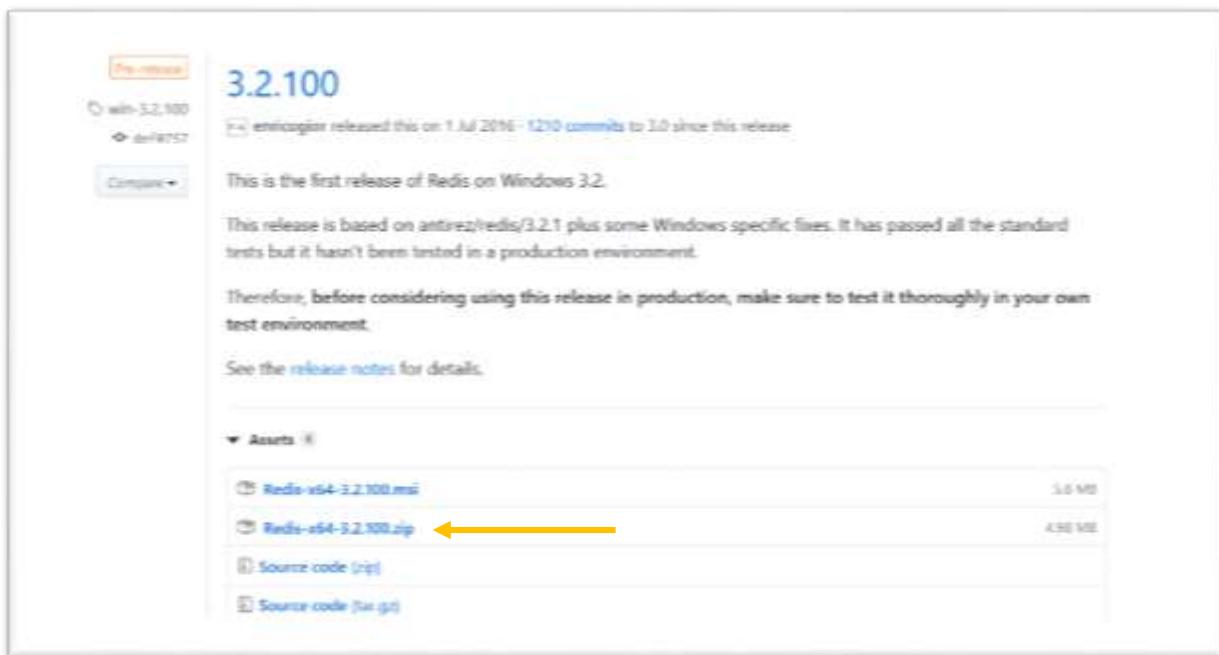
Сделать запрос всех данных в таблице транс:

```
$ curl -s -H 'диапазон: строки=0-2' http://localhost:8085/trans.
```

Для подробного ознакомления с тем, как использовать все сервисы с поддержкой RESTful (запрос, вставка, обновление и/или удаление данных), ознакомьтесь с **Приложением "Restful SQLite GET/POST commands"**.

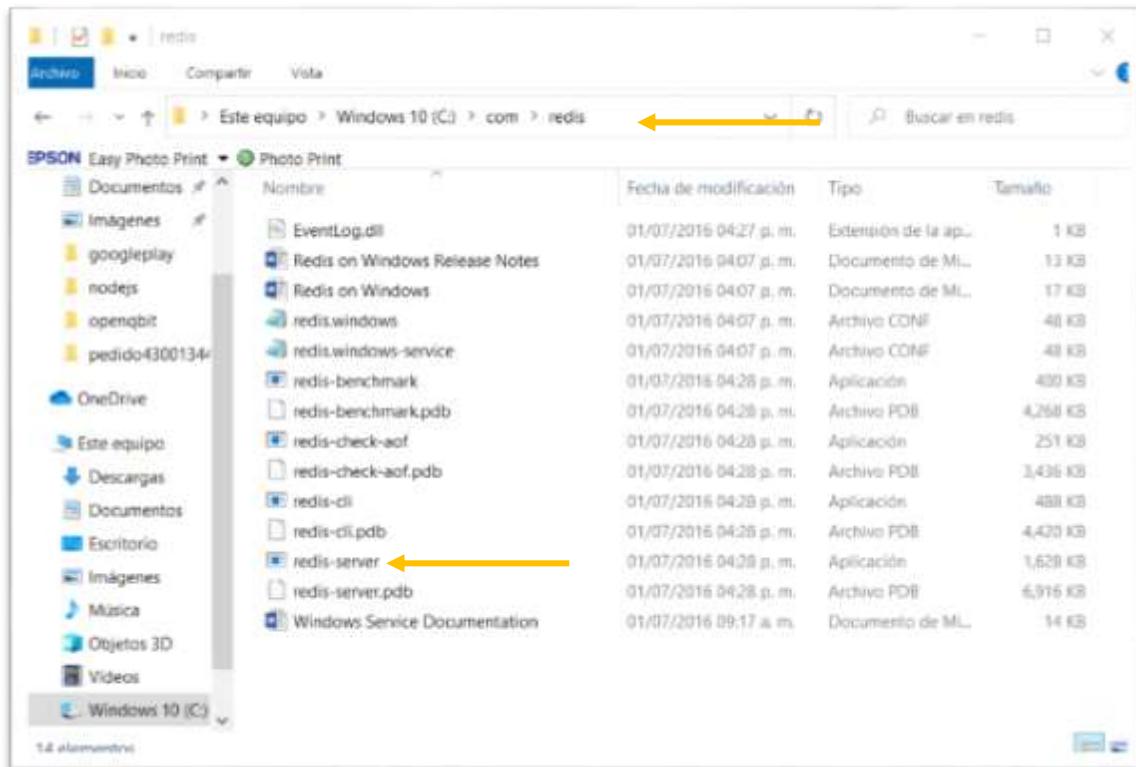
ПРИМЕЧАНИЕ: В предыдущей установке все запросы делались по URL с адресом "localhost", но когда сервер открыт с публичным IP (Интернет) или частным IP (Wifi), он будет работать без проблем. Мы протестируем это при выполнении тестов связи между сервисом SQLite RESTful и узлами сети связи, которые образуют Mini BlocklyChain.

Установка базы данных Redis для службы сети резервного копирования, для загрузки программы Redis для Windows нужно перейти на сайт <https://github.com/microsoftarchive/redis/releases/tag/win-3.2.100> и выбрать пакет ZIP.



Чтобы найти установку, мы создадим директорию под названием "redis" в Windows и загрузим файл с расширением ZIP, распакуем его в директорию, созданную ранее, у нас уже завершена установка redis.

Мы тестируем установку, запустив сервер двойным щелчком на команде "redis-server".



После выполнения команды "redis-server" мы увидим сервер, работающий в командном терминале Windows CMD на стандартном порту 6379:

```
C:\com\redis\redis-server.exe
[10192] 31 May 13:50:45.348 # 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
```

В этом тесте у нас есть версия Redis 3.2.100 для Windows 10, в случае операционной системы Linux у нас есть версия 6.0.4 (выпущена в мае 2020 года), однако для нашей конфигурации Mini BlocklyChain версия Windows имеет достаточно функций, которые нам нужны.

Чтобы остановить выполнение, мы делаем комбинацию клавиш Ctrl + C. Далее мы настраиваем базу данных Redis 3.2.100 для Windows 10 с конфигурацией между узлами сети связи Redis и Mini SQLSync, тип **Master(Master)-Slave(Slave)** распределяется следующим образом:

Мастер - это сервер Redis для Windows 10, который мы настраиваем, и он будет реплицировать данные в режиме реального времени и синхронизироваться с одной и той же информацией на все узлы (мобильные телефоны). На этих узлах будет установлен сервер Redis в режиме **ведомого устройства**.

На данном этапе мы начинаем видеть, как работает сеть резервного копирования, которую мы устанавливаем и настраиваем. Такая конфигурация послужит нам для связи со всеми узлами в режиме реального времени, она поможет нам общаться со всеми узлами сети при появлении новой очереди транзакций, обрабатываемой узлами, наравне с передачей информации всем узлам, так что любой узел будет иметь одинаковую вероятность обработки информации об "очереди транзакций".

Начинаем настройку коннектора **SQLite-Redis Sentinel**.

Этот коннектор представляет собой программу, разработанную на языке Java и, как следует из его названия, соединяющую базы данных SQLite и Redis (**Master**).

Функция коннектора заключается в передаче информации (транзакций) из SQLite в Redis (**Мастер**), и он посыпает "очередь транзакций" узлам (**Slaves**).

Подробнее о Java-коде коннектора **SQLite-Redis Sentinel** см. в приложении "Коннектор SQLite-Redis Java-кода".

Конфигурация базы данных Redis (**Master**) **redis.conf** для Windows 10.

Добавьте следующие изменения или директивы в файл, сохраните изменения и запустите сервер Redis

Начните с того, что найдете настройку **tcp-keepalive** и установите ее на 60 секунд, как предложено в комментариях. Это поможет Redis обнаружить проблемы с сетью или обслуживанием:

tcp-keepalive 60

Найдите директиву **requirepass** и сконфигурируйте ее с сильной ключевой фразой. В то время как ваш трафик Redis должен быть защищен от третьих сторон, это обеспечивает аутентификацию Redis. Так как Redis быстрый и не оценивает попытки использования пограничной парольной фразы, выберите сильную, сложную парольную фразу для защиты от попыток грубой силы:

requirepass тип_your_network_master_password

Пример:

requirepass FPqwedsLMdf76ass7asddfd2g45vBN8ty99

Наконец, есть некоторые опциональные параметры, которые вы, возможно, захотите настроить в зависимости от сценария использования.

Если вы не хотите, чтобы Redis автоматически добавлял самые старые и наименее используемые клавиши по мере заполнения, вы можете отключить автоматическое удаление клавиш:

гамма-память-политика

Для повышения гарантии долговечности можно включить дополнительную сохраняемость файлов. Это позволит минимизировать потери данных в случае сбоя системы за счет больших файлов и немного более низкой производительности:

дополнительно да

аппендульм "redis-staging-ao.aof"

Приступите к сохранению изменений и перезапустите службу Redis для Windows 10, остановитесь с помощью клавиш Ctrl + C и снова запустите командную строку Windows CMD:

C: \redis_redis_directory redis_server

В нашем примере мы видим, что у нас подключен (**ведомый**) узел.

```
:\\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
inally set to 1024).

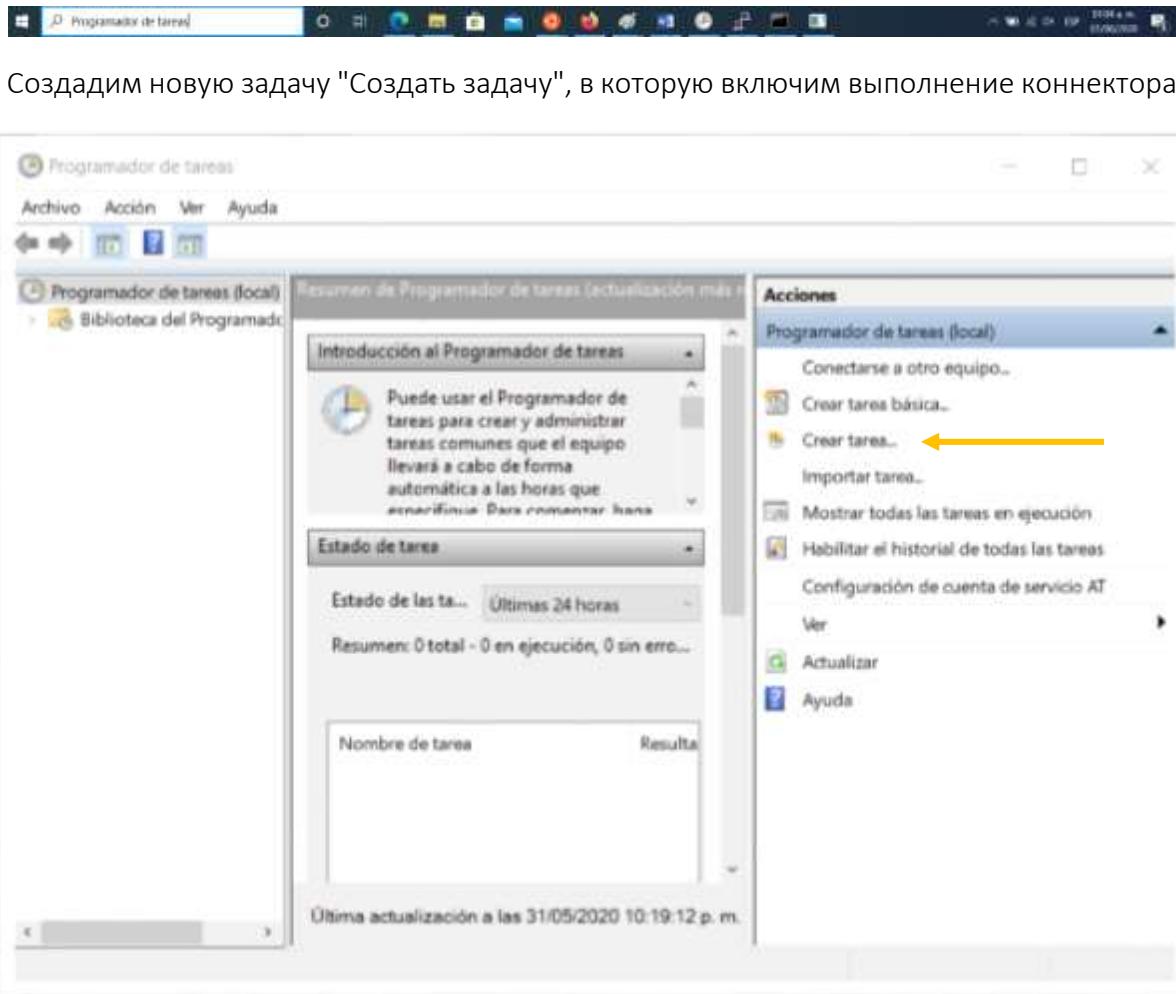
Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 51

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
```

Мы видим, что мы синхронизировали узел с IP 192.168.1.68 в стандартном порту 6379.

Теперь нам нужно запланировать в операционной системе Window 10 задачу автоматического выполнения коннектора **SQLite-Redis Sentinel**. Мы делаем это с помощью инструмента Windows 10 - мы выполняем его снизу слева, набирая "Планировщик задач".

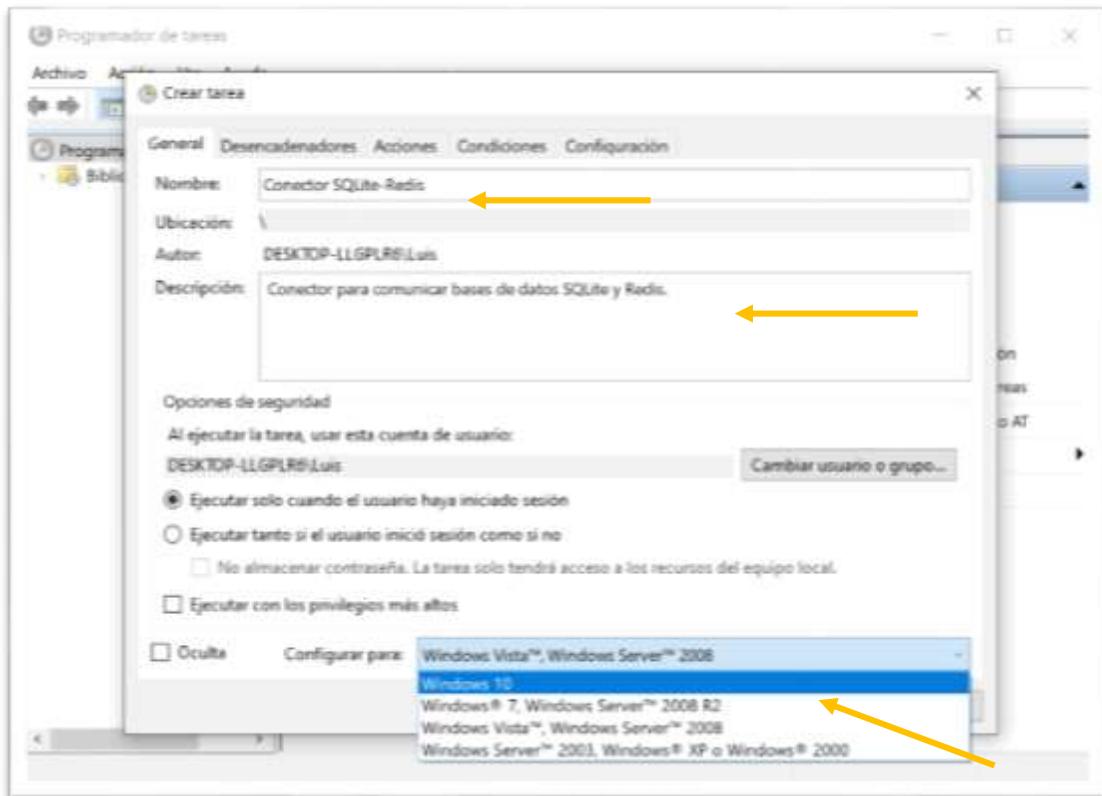


При нажатии на кнопку "Создать задание" откроется дополнительное окно, где на вкладке "Общие" необходимо задать следующие параметры:

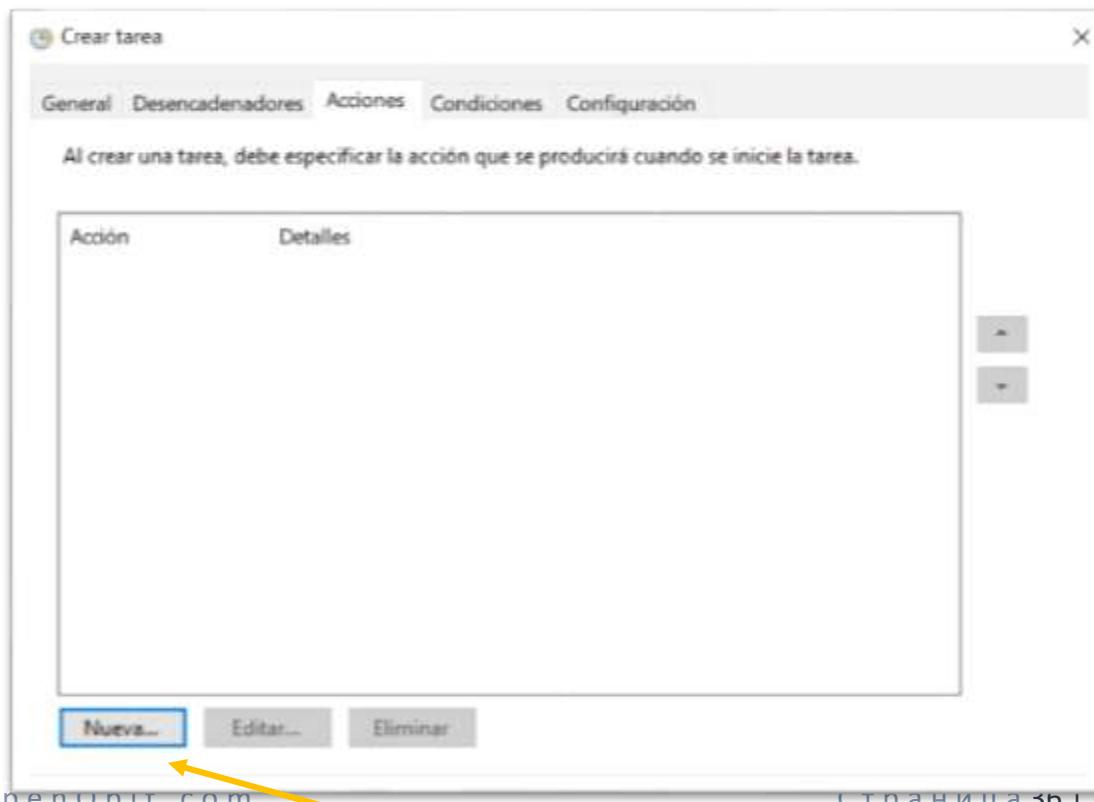
Имя: task_name

Описание: необязательный

Настройка для: select_operating_system_windows_version



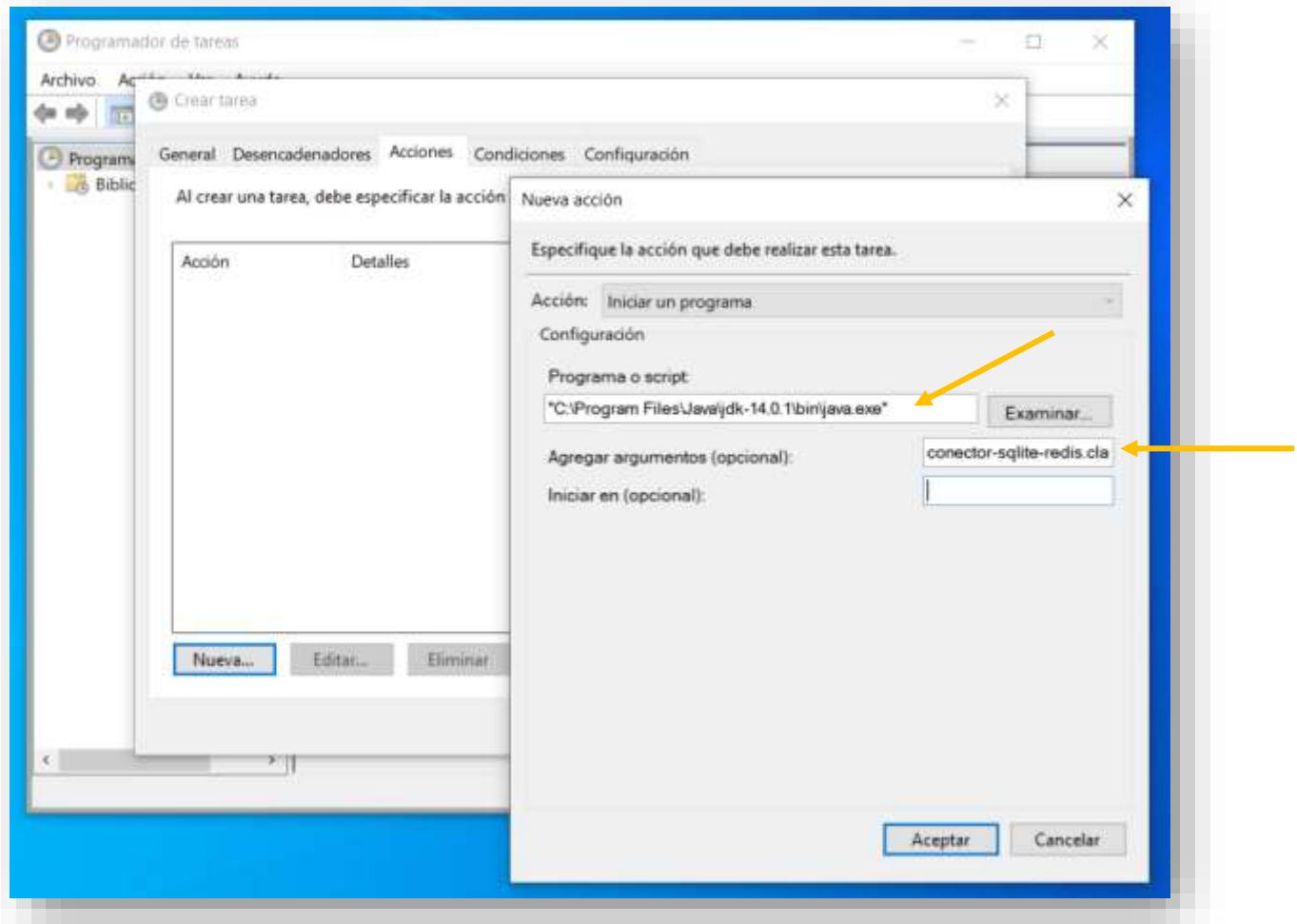
Изменить, перейдя на вкладку "Действия", и нажать кнопку "Создать":



Мы даем следующие параметры:

Программа или скрипт: коннектор_путь

Добавление аргументов: имя коннектора



Вышеуказанные параметры могут отличаться в зависимости от расположения разъема. Основной идеей является выполнение программы **коннектора-qlite-redis-v1.class**, как это обычно делается в командной строке:

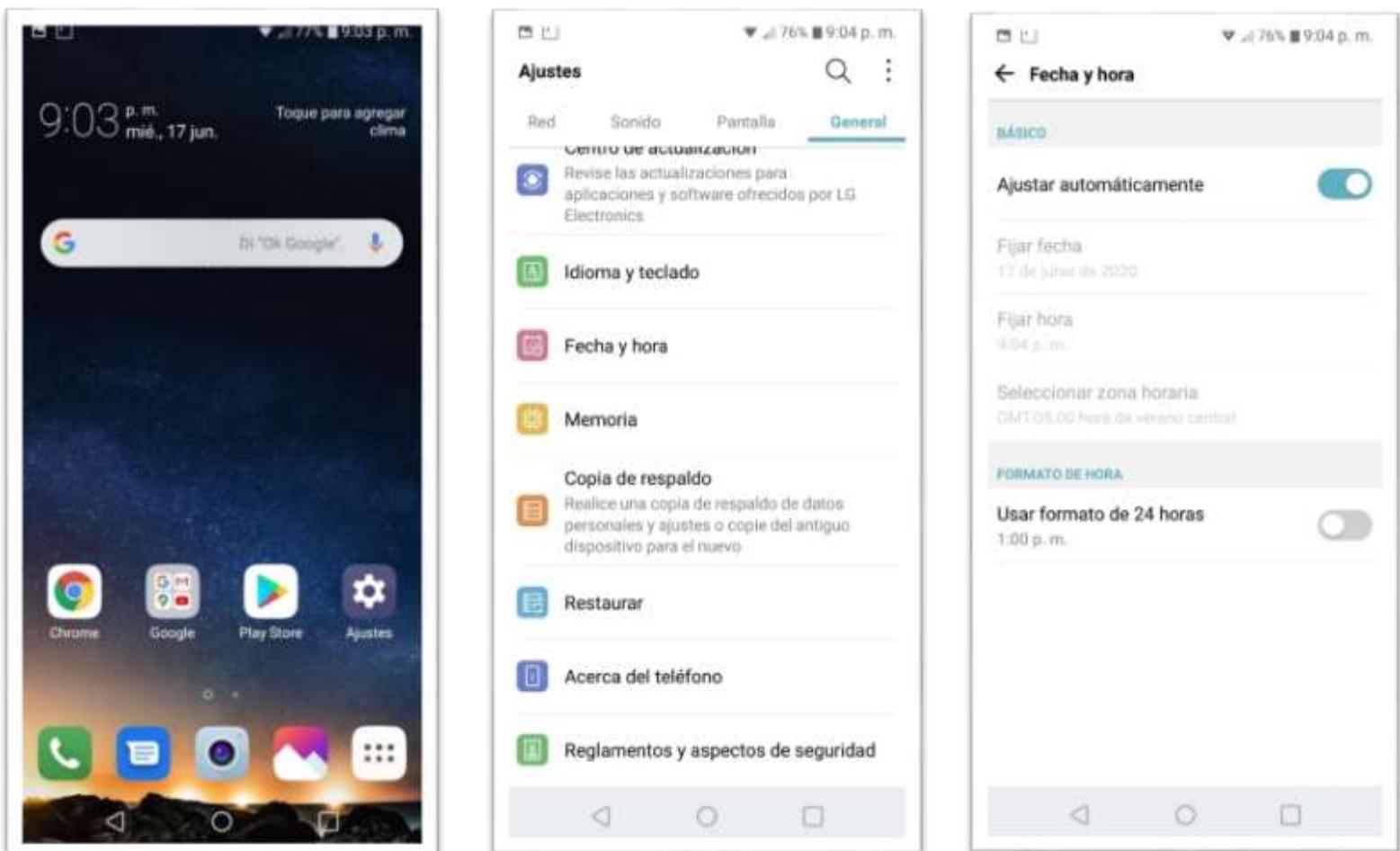
C: \jdk_директория java-коннектор-sqlite-redis-v1

Для завершения необходимо выбрать закладку "Триггеры", в которой мы дадим параметры того, когда (день, час, минуты) мы хотим, чтобы задача была выполнена, эти параметры основаны на бизнес-правилах, которые необходимо будет создать в системе Mini BlocklyChain.

10. Синхронизация в узлах системы (мобильный телефон) минуты и секунды.

Очень важно, чтобы все узлы синхронизировались в основном в минутах и секундах. Так как при отправке или публикации в определенное время очереди транзакций все узлы должны быть синхронизированы, так как это будет зависеть от менеджера задач, который будет создан в локальной системе с помощью инструмента CRON, который будет выполняться одновременно во всех узлах, то это сделает так, что все узлы с одинаковой вероятностью получат право быть выбранными, чтобы иметь возможность обрабатывать очередь транзакций и генерировать новый блок для добавления его в цепочку блоков системы. Для синхронизации узлов у нас есть два варианта.

Первый вариант синхронизации узла, мы сможем сделать это простым способом. В нашем устройстве через внутреннюю опцию, которая включает в себя систему Android, мы должны перейти к части **Настройки > Дата и время > Настройка автоматически**



С предыдущей конфигурацией мы будем синхронизировать в минуты и секунды все узлы системы независимо от того, какая страна в мире уже синхронизирована основана на каждой географической области возможно, какие изменения это время, но в зависимости от минут и секунд должно быть синхронизировано это достаточно для нас, чтобы запустить процесс задач на плановой основе с инструментом cron во всех узлах, чтобы запустить каждый определенный момент времени в минутах, т.е. мы можем создать задачу в crontab для запуска каждые 10 минут или 30 минут в зависимости от каждого дизайна системы.

Это будет полезно при использовании сети связи "Peer to Peer", в случае использования резервной сети этот процесс не будет применяться, так как распределение очереди транзакций производится по модели клиент-сервер и именно сервер управляет инструментом cron.

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

Вторым вариантом является использование внешнего API, где мы будем выполнять команду Curl через расширение (**ConnectorSSHClient**).

Место, где мы будем использовать внешние службы NTP (Network Time Protocol) - это место, где мы будем использовать внешние службы NTP (Network Time Protocol):

<http://worldtimeapi.org/>

Теперь мы рассмотрим способ получения времени с NTP-серверов, расположенных по всему миру, и это поможет нам получить все узлы для запроса в определенное время в одну и ту же дату и время.

Пример запроса с расширением (**ConnectorSSHClient**).

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

Мы осуществили подключение к терминалу Termux:



Мы выполняем команду Кёрла:



Нужно учитывать, что результат команды Curl будет в формате JSON, что-то похожее на следующий результат:

```
#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
```

Также результат может быть в формате JSON без данных, отформатированных в или в линейном формате JSON, как показано ниже:

```
{"abbreviation": "CDT", "client_ip": "200.77.16.151", "datetime": "2020-06-18T14:19:07.216800-05:00", "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": 1592507947, "utc_datetime": "2020-06-18T19:19:07.216800+00:00", "utc_offset": "-05:00", "week_number": 25}.
```

Любой из двух предыдущих способов нам придётся фильтровать информацию через существующее расширение JSON типа "JSONTOOLS" или использовать фильтры в App Inventor при обработке текста, чтобы получить только час, день или секунды в соответствии с потребностями каждой системы. После обработки результата можно провести логическое сравнение, и на основе этого сравнения можно выполнить задачу, уже запрограммированную с помощью сервиса "cron", конфигурацию которого мы увидим позже в каждом узле.

Ссылка на расширение JSONTOOLS:

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

Теперь мы рассмотрели два варианта синхронизации времени узлов. Мы продолжим настройку сервиса "cron" на каждом узле.

Конфигурация автоматизированного планирования задач для выполнения с помощью сервиса **CRON** на системах Android (узлах системы).

Сначала нужно понять, как работает автоматический планировщик.

Сервис cron обычно все системы имеют эту предварительную установку, и где все задачи, которые будут выполняться автоматически, запланированы, находятся в файле, называемом crontab.

Редактируем кронтаб-файл с **\$ crontab -e**, используем **Vi** редактор, внутри используем формат:

```
# m h dom mon dow user command
```

Где:

- **m** соответствует минуте, в которую будет выполнен скрипт, значение идет от 0 до 59
- **h** точное время, формат - 24 часа, значения - от 0 до 23, значение - 0 12:00 полуночи.
- **dom** относится к дню месяца, например, вы можете указать 15, если хотите запускать каждые 15.
- **dow** означает день недели, он может быть числовым (от 0 до 7, где 0 и 7 - воскресенье) или первыми 3 буквами дня на английском языке: mon, tue, wed, thu, fri, sat, sun.
- **пользователь** определяет пользователя, который будет выполнять команду, это может быть root или другой пользователь, если у него есть разрешение на выполнение сценария.
- **команда** ссылается на команду или абсолютный путь выполняемого скрипта, например: /home/user/scripts/update.sh, если вы вызываете скрипт, то он должен быть исполняемым

Чтобы прояснить ситуацию, объясните несколько примеров задач, стоящих перед cron:

```
15 10 * * * пользователь /home/user/scripts/update.sh  
Вы будете запускать скрипт update.sh каждый день в 10:15 утра.
```

```
15 22 * * * пользователь /home/user/scripts/update.sh  
Вы будете запускать скрипт update.sh каждый день в 22:15.
```

```
00 10 * * 0 root apt-get - и обновление User root  
Вы будете запускать обновление каждое воскресенье в 10:00 утра.
```

45 10 * * Солнечный корень apt-get – и обновление
Корневой пользователь будет запускать обновление каждое воскресенье (солнце) в
10:45 утра.

Мы сохранили изменения в редакторе и на этом закончили настройку сервиса cron.
Если у вас в системе не установлен cron, вы можете сделать это с помощью следующей
команды:

\$ apt install cron

2. Установка и настройка сетевых узлов - Мобильные телефоны.

Начнем с сети связи для узлов, которые будут использовать Mini BlocklyChain.

Сначала нам нужна среда Linux, так как каждая система Android основана на Linux для безопасности и гибкости инструментов, мы будем использовать терминал "Termux", который содержит ту среду, в которой мы будем устанавливать коммуникационную сеть.

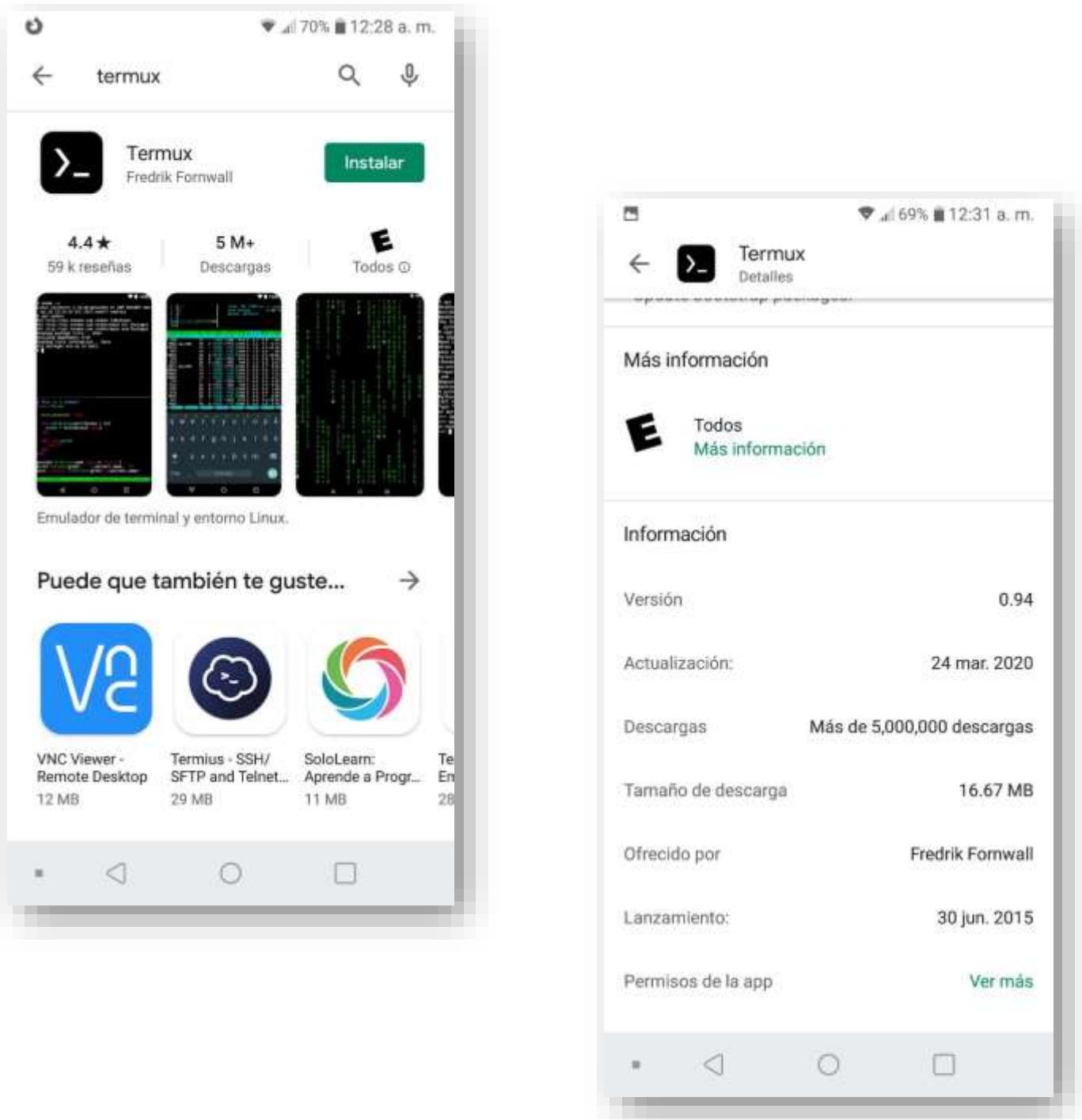
Termux - это эмулятор Linux, на котором мы установим необходимые пакеты для создания нашей коммуникационной сети между узлами.

Одним из главных преимуществ использования терминала Termux является то, что вы можете устанавливать программы без необходимости "поворачивать" мобильный телефон (смартфон), что гарантирует, что гарантия производителя не будет потеряна в связи с данной установкой.

Установка терминала. Со своего мобильного телефона перейдите в приложение с иконкой Google Pay (play.google.com).



Выполните поиск по приложению "Termux", выберите его и запустите процесс установки.



Запуск приложения Termux.

После запуска нам придется выполнить следующие две команды для выполнения обновления эмулятора операционной системы Linux:

\$ соответствующее обновление

\$ подходящий апгрейд

Подтвердите все опции Y(Yes)....

Termux

Home \$ apt обновление

\$ apt апгрейд

```
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
$
```

```
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
$ apt update ←
```

```
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
$ apt upgrade ←
```

11. Конфигурация хранения в Termux.

После того, как вы обновили и модернизировали систему Termux, мы начнем настройку просмотра внутреннего хранилища телефона в системе Termux. Это поможет вам иметь

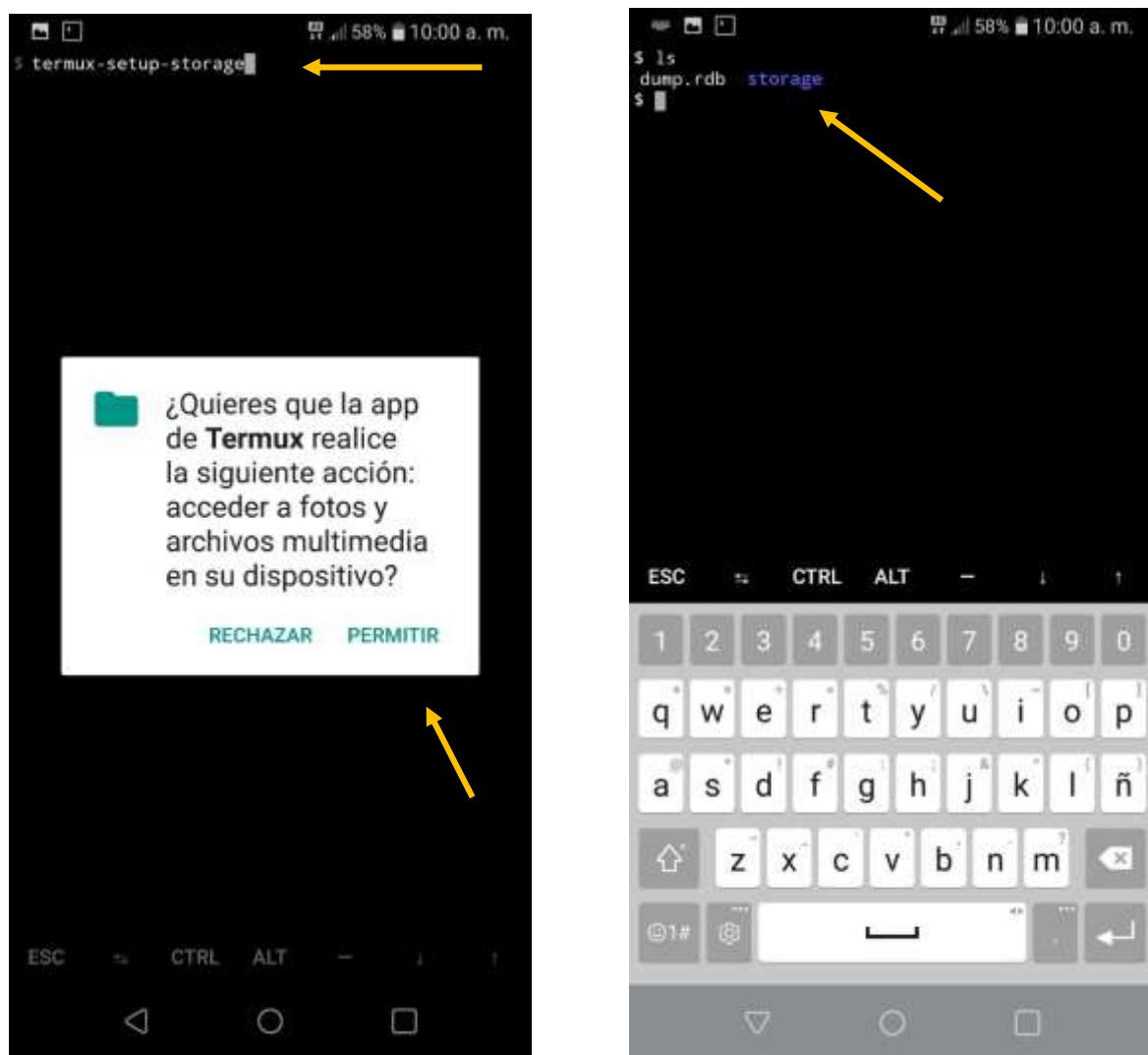
возможность обмениваться информацией между Termux и нашей информацией в телефоне.

Это можно сделать просто и быстро, выполнив следующую команду на терминале Termux.

\$ срочное хранение

При выполнении предыдущей команды появляется окно с просьбой подтвердить создание виртуального хранилища (каталога) в Termux. Мы проверяем, отдавая команду:

\$ ls



12. Установка сети "Tor" и "Syncthing".

\$ apt install tor

\$ подходящая установка синхронизации

Принять установку, набрав заглавную букву Y в обоих случаях по запросу....

\$ 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, 2 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
ges-24 stable/main arm syncthing arm 1.5.0 [6407
kB]
27% [1 syncthing 2193 kB/6407 kB 34%]
```



13. Установка базы данных "Redis" и сервера SSH (Secure Shell).

\$ подходящая установка redis

\$ подходящая установка openssh

\$ подходящий монтажный путепровод

\$ apt install redis

\$ apt install openssh

\$ apt install sshpass

```
$ 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
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
kes-24 stable/main arm libdb5 arm 18.1.32-4 [465
kB]
Get:2 https://dl.bintray.com/termux/termux-pac
kes-24 stable/main arm krb5-libs arm 1.18.1 [839
kB]
24% [2 krb5-libs 131 kB/839 kB 16%]
```

```
$ 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]
```

Мы закончили установку сети связи, продолжаем настройку пакетов: Tor, Syncthing и Redis DB.

14. Настройка сервера SSH на мобильном телефоне (смартфоне).

Мы дадим возможность SSH-серверу в мобильном телефоне подключаться с нашего ПК к мобильному и работать быстрее и комфортнее, а также это послужит для проверки правильности работы сервиса SSH-сервера в мобильном телефоне, так как мы будем использовать его в сети связи в Mini BlocklyChain.

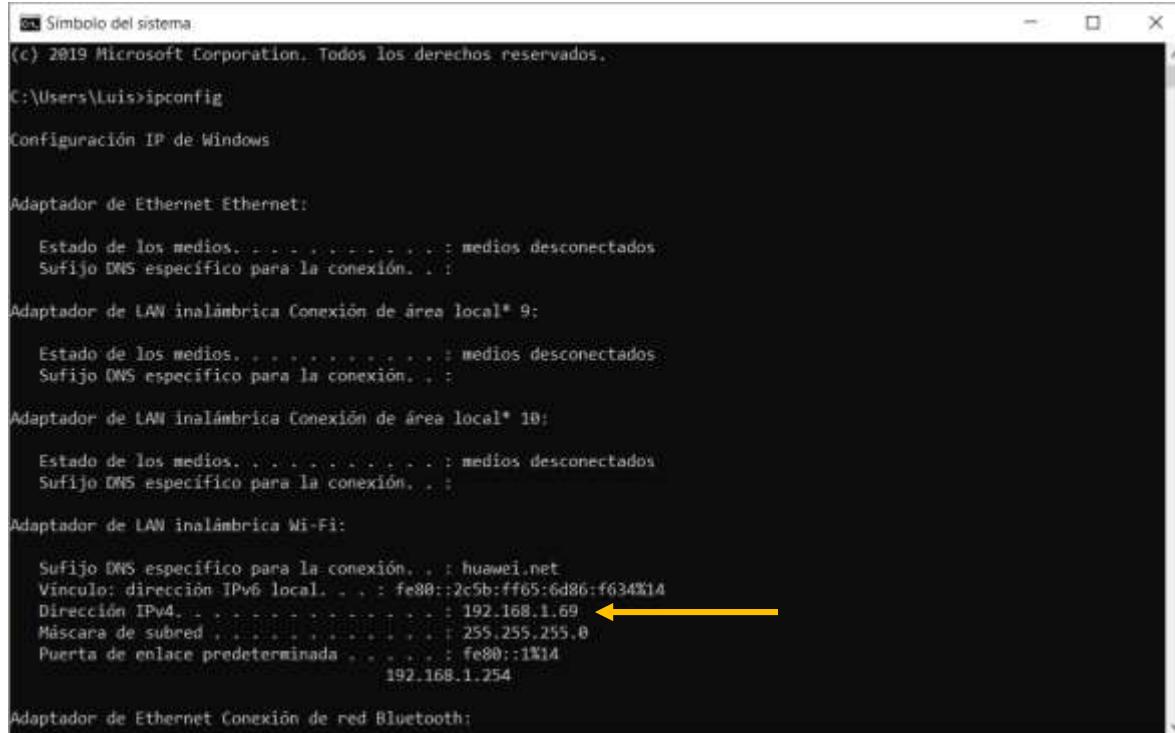
Первое, что нам нужно сделать, это подключить мобильный и компьютер к **одной и той же WiFi сети**, чтобы они могли видеть друг друга. IP-адреса или адреса должны быть похожи на 192.168.XXX.XXX Значения XXX - это переменные числа, которые присваиваются случайным образом в каждом компьютере.

Этот пример был протестирован на мобильном телефоне LG Q6 и ПК с Windows 10 Home.

Проверьте IP или адрес, который ПК подключил к WiFi, мы должны открыть терминал в Windows.

На нижней панели, где находится поисковая лупа, напишите cmd и нажмите клавишу Enter. Откроется терминал и в нем мы напишем команду:

C:\Имя пользователя> ipconfig



```
Símbolo del sistema | Símbolo del sistema
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Luis>ipconfig

Configuración IP de Windows

Adaptador de Ethernet 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
  Vinculo: 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:
```

Он покажет нам IP-адрес, присвоенный ПК в этом файле - 192.168.1.69, но, скорее всего, он будет разным в каждом случае.

ЗАМЕЧАНИЕ: Адрес, на котором написано "IPv4 адрес" должен быть взят, чтобы его не путать со шлюзом.

Теперь в случае с мобильным телефоном в терминале Termux мы должны ввести следующую команду, чтобы узнать имя нашего пользователя, которое мы будем использовать для подключения к SSH-серверу, у которого есть наш телефон, мы выполняем следующую команду:

Теперь в случае с мобильным телефоном в терминале Termux мы должны ввести следующую команду, чтобы узнать имя нашего пользователя, которое мы будем использовать для подключения к SSH-серверу, у которого есть наш телефон, мы выполняем следующую команду:

\$ whoami

Позже мы должны дать пароль этому пользователю, поэтому мы должны выполнить следующую команду:

\$ pas

Он попросит нас ввести пароль и нажмет Enter, снова попросит нас ввести пароль, который мы подтверждаем и нажмет Enter, если он был **успешно "Новый пароль был успешно установлен"**, в случае пометки ошибки возможно, что пароль был набран неправильно. Выполните процедуру еще раз.

А затем, чтобы узнать, какой IP у нас есть в Termux, мы вводим следующую команду, IP после слова "**inet**":

\$ ifconfig -a

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



```
e:0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST
> mtu 1500
      inet 192.168.1.68 netmask 255.255.255.0
      broadcast 192.168.1.255
      inet6 fe80::c1ff:fee6:3051 prefixlen 64
      scopeid 0x20<link>
      unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
      RX packets 908745 bytes 947916536 (904.0 MiB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 601034 bytes 93496881 (89.1 MiB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
$ 
```

Теперь пришло время запустить службу SSH-сервера на вашем телефоне, чтобы вы могли получать сеансы с вашего ПК. Мы выполняем следующую команду в терминале Termux, эта команда не дает никакого результата.

шшд

```
$ sshd
$ 
```

Теперь нам придется установить на ПК программу, которая будет взаимодействовать с SSH-сервером телефона с ПК.

Надо зайти на <https://www.putty.org>.

Выберите, где находится ссылка "Вы можете скачать PuTTY здесь".



Download PuTTY

PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the Windows platform. It is released under the MIT License, 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 recommendations.



Bitvise SSH Client

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

- graphical SFTP file transfer;
- single-click Remote Desktop tunneling;
- 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](#).

Выбирайте 32-битную версию, не имеет значения, будет ли ваша система 64-битной или нет.

Download PuTTY: latest release

[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 see 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")

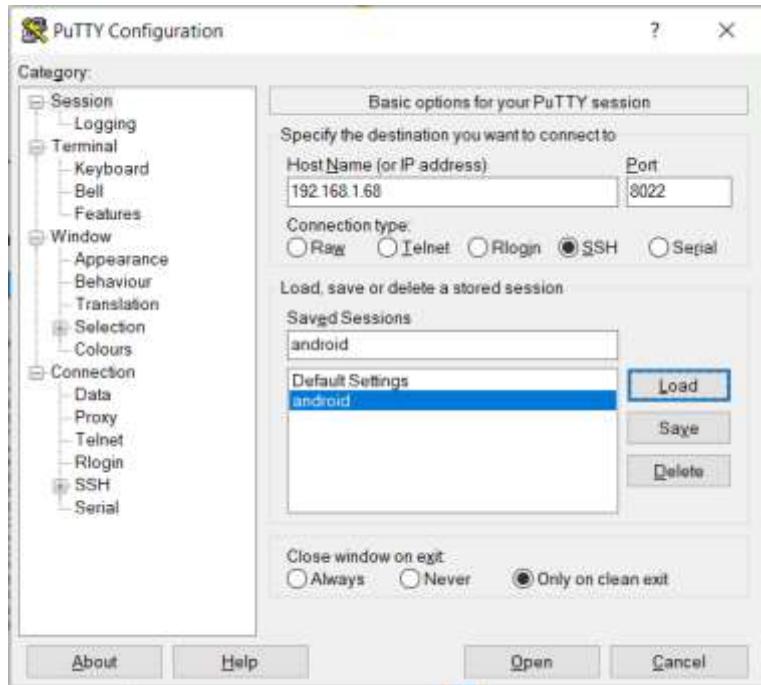
32-bit:	putty-0.73-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.73-installer.msi	(or by FTP)	(signature)

Unix source archive

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

После загрузки на компьютер запустите его и установите с параметрами по умолчанию. Затем запустите приложение PuTTY.

В этой сессии мы введем данные с нашего Openssh сервера, который мы установили в мобильный телефон.



Введите IP-адрес мобильного телефона.

Имя хоста или IP-адрес:

192.168.1.68 (пример IP)

Порт:

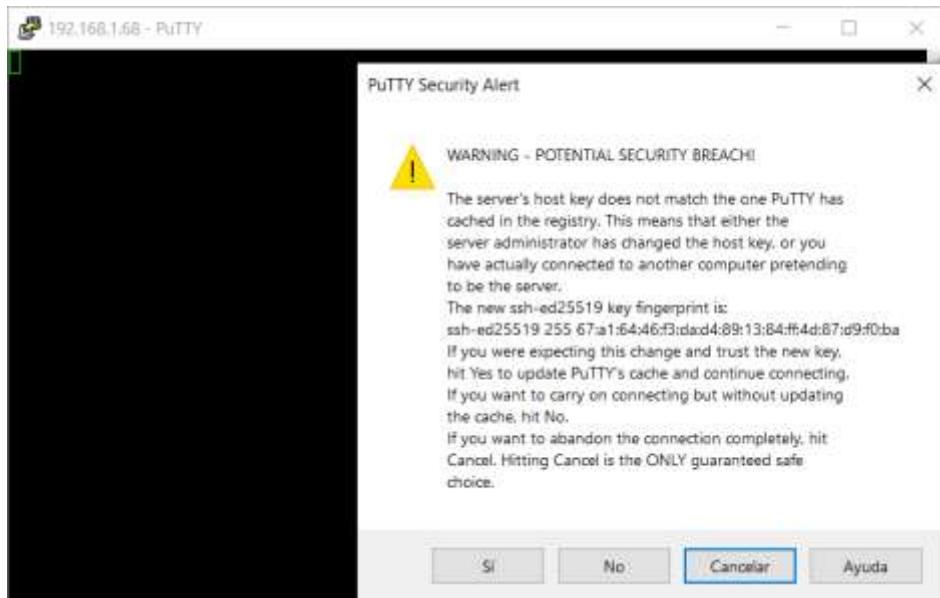
8022 (порт по умолчанию мобильного SSH-сервера).

Мы можем дать название сессии в разделе "Сохраненные сессии" и

нажать кнопку "Сохранить".

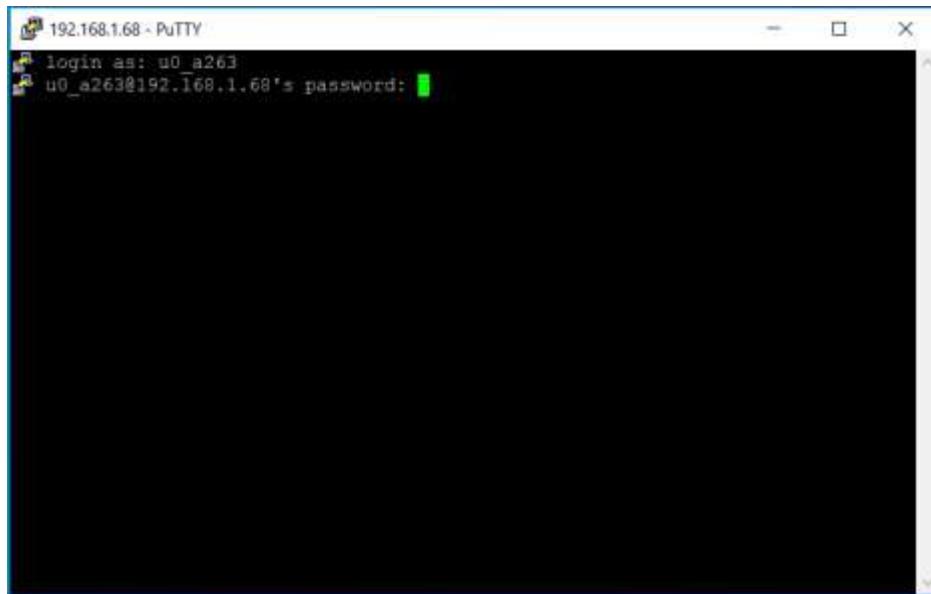
Позже в нижней части мы нажимаем, чтобы открыть соединение с сервером, давая кнопку "Открыть".

При первом **подключении** к компьютеру вас **попросят подтвердить** ключ шифрования информации, нажав кнопку "Да".



Позже нас спросят о пользователе, с которым мы собираемся связаться. Мы будем использовать ранее полученную информацию (пользователь и пароль).

Вход как: мы должны ввести нашего пользователя и дать Enter, затем мы попросим пароль еще раз дать Enter.



Если данные были верны, то мы находимся в сеансе SSH (Secure Shell), выполненном с компьютера (клиента) на телефоне (SSH Server).

```
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

$
```

ВАЖНОЕ ЗАМЕЧАНИЕ: Помните, что IP (адрес) компьютера и IP (адрес) мобильного телефона, подключенного к одному и тому же WiFi, вероятно, будет меняться каждый раз, когда мы отсоединяемся и переподключаемся, поэтому мы должны дважды проверить, какие адреса есть у каждого устройства, это обеспечит успех соединения между устройствами через SSH-сервер телефона и ПК (Клиент).

До сих пор мы могли подключаться только к той же сети WiFi, но если мы переместим наш телефон за пределы той же сети, что и компьютер, мы не сможем подключиться, потому что есть разные сети, вовлеченные в работу с другими, более сложными устройствами связи. Мы решим эту проблему, когда будем настраивать сеть "Tor".

Помните, что это подключение производится только для того, чтобы проверить работу сервера, который мы установили на телефоне, и иметь более комфортную рабочую среду с удаленной сессией с ПК на телефон.

15. Сетевая конфигурация "Tor" со службой SSH (Secure Shell).

С помощью удалённой сессии с ПК мы начнем конфигурирование сети "Tor" с включённым сервисом SSH.

Важность наличия сети "Tor" заключается в том, чтобы дать устройствам возможность общаться в любой точке мира через Интернет, не находясь в одной WiFi сети, независимо от того, где мы находимся, мы сможем подключаться и формировать сеть "Peer to Peer" между узлами, что является существенной функциональностью архитектуры Mini BlocklyChain.

Сеть "Tor" обладает большой гибкостью в настройке, т.к. в конфигурационном файле "torrc" есть несколько параметров, по которым этот файл находится в пути (`$PREFIX/etc/tor/torrc`) в нашем случае с сеанса Termux с нашего ПК мы узнаем, набрав следующую команду:

```
$ эхо $PREFIX
```

```
/data/data/com.termux/files/usr
```

Поэтому файл, который нам нужно отредактировать, будет находиться в пути:

```
PREFIX/etc/tor/torrc, что равно /data/data/com.termux/files/usr/etc/tor/torrc
```

Редактируем конфигурационный файл с помощью редактора командной строки "vi", выполняя следующую команду:

vi /data/data/com.termux/files/usr/etc/tor/torrc

Он отредактирует этот файл для нас, в качестве примера:



Отредактированный файл "torrc":

```

192.168.1.68 - PuTTY
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 0%

```

В этом "торковом" файле мы должны будем добавить или использовать строки, которые есть в файле, внеся следующие изменения, три строки:

Синтаксис: SOCKSPort <номер порта приложения>

Пример: SOCKSPort 9050

Переменная **SOCKSPort** сообщает нам, что данное коммуникационное сокет по протоколу TCP-IP будет использовать мобильное устройство (телефон) по умолчанию и порт 9050 будет открыт или использован.

Синтаксис: HiddenServiceDir <Каталог, в который будет сохранена конфигурация приложения>.

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

Переменная **HiddenServiceDir** сообщает нам, что это будет каталог, в котором будет храниться конфигурация сервиса, который будет использоваться через сеть Tor. В этом

каталоге вы найдете файл конфигурации и безопасности для сервиса, а в этом каталоге вы найдете файл под названием **hostname** (**имя хоста**).

Мы можем видеть адрес, назначенный сетью Тор для конкретного сервиса, созданного в нашем случае - это создание SSH-сервиса, который будет реализован в сети Тор, каталог, который мы называем **hidden_ssh**, будет содержать файл с **именем хоста**. Этот каталог не нужно создавать, потому что при запуске сетевого сервиса Тор он будет создан автоматически.

Чтобы увидеть адрес, предоставленный сетью Тор, мы можем использовать команду "more". Перед использованием этой команды мы должны закончить настройку файла "torrc" и зарегистрировать сетевой сервис Тор так, чтобы был создан каталог **hidden_ssh** и файлы внутри него, как в случае с файлом **с именем хоста**.

больше /данные/данные/ком.termux/файлы/

В результате выполнения вышеприведенной команды будет получен адрес с буквенно-цифровой строкой с расширением :

uqwwf6ojdmoybyzvudoq3x2weq7k7rjitblhba3pjba2nvjmx3wer.onion

Наконец, нам нужно добавить переменную **HiddenServicePort** в конфигурационный файл "torrc". Этот параметр сообщает сети Тор, какой порт будет использовать приложение, на которое мы подписываемся, по сети Тор.

Синтаксис: **HiddenServicePort <Порт отправления> <Локальный или публичный IP>: <Порт отправления>**.

о

HiddenServicePort <Порт отправления>

Примеры:

HiddenServicePort 22 127.0.0.1:8022

о

HiddenServicePort 8022

С вышеизложенным мы закончили настройку сети Tor для общих служб и службы SSH (Secure Shell), которая будет использоваться для связи обновления базы данных SQLite, где мы сохраним процессы валидации нашей системы Mini BlocklyChain.

Мы продолжаем настройку сети связи Mini BlocklyChain.

16. Пиринговая конфигурация системы с ручной синхронизацией.

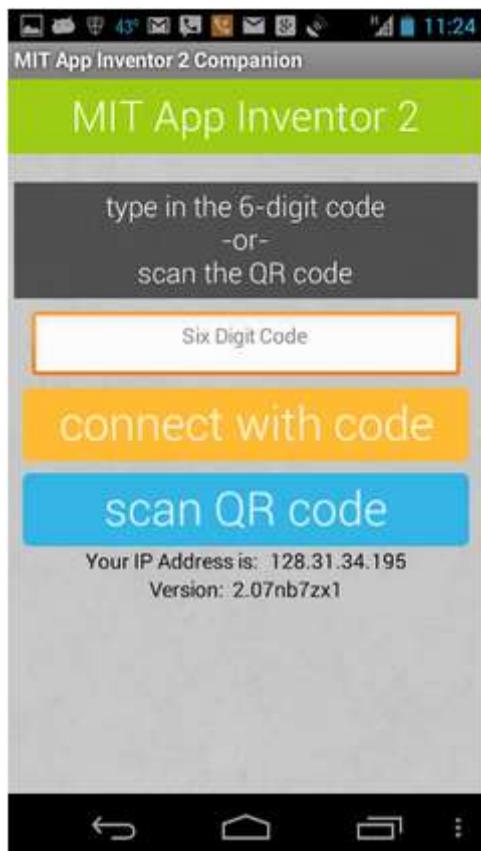
Архитектура "Peer to Peer" является фундаментальной в системе блочно-цепных технологий, поскольку эта архитектура дает две центральные точки в процессах системных коммуникаций, одна из которых заключается в предоставлении одной и той же обновленной (синхронизированной) информации в любое время на всех узлах, независимо от того, находятся ли узлы в частной сети (Wifi) или в сети общего пользования (Интернет), или же гибрид этих двух и второй точкой такого типа архитектуры является то, что они не зависят от посредника (сервера) в передаче, обновлении или консультировании информации между узлами. Все это связано с тем, что связь осуществляется непосредственно между узлами, в нашем случае Mini BlocklyChain связь осуществляется непосредственно между телефонами, которые формируют сеть без посредника.

Для этой задачи мы будем использовать инструмент синхронизации с открытым исходным кодом, который работает на основе архитектуры "Peer to Peer".

Конфигурация SyncThing для устройств (мобильных телефонов) будет отображаться вручную и автоматически. Ручная форма применяется при синхронизации до 5 узлов, в случае наличия большого количества автоматических конфигураций является оптимальным, процесс фокусируется на регистрации узлов, с которыми мы хотим выполнить синхронизацию выбранной информации.

Требования для начала:

1. Запустили службу сети "Tor".
2. (опционально - рекомендуется) Установив приложение, способное считывать QR-коды, мы предлагаем приложение App inventor Android, которое легко и просто установить из Google Play, а позже в этом руководстве мы будем использовать его в разделе "Определение и использование блоков в мини-блоклейке".
3. Чтобы инициировать службу Синтинга.



Мы показываем приложение App Inventor, которое мы будем использовать для считывания QR-кодов, которые будут служить нам в будущем для регистрации узлов в Syncthing.

Следующий пример был приведен между двумя мобильными устройствами (телефонами) с использованием следующих моделей:

Мобильное устройство №1: Модель LG Q6

Мобильное устройство №2: модель Сансумг

Мы начнем с запуска сетевых сервисов Тор и синхронизации сервисов с помощью следующих команд, откроем два терминала внутри Termux и выполним каждую команду по отдельности:

\$ tor

После того, как Вы напечатали команду запуска сетевой программы Тор, Вы должны проверить, что выполнение прошло успешно, проверив, что оно было выполнено на 100%.

Запустив программу Тор на терминале Termux, мы проверяем, что она работает на 100%.

\$ top

```

$ 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

```

ESC ≈ CTRL ALT - + ↑ ↓

* < ○ □

```

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): Starting
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

```

ESC ≈ CTRL ALT - + ↑ ↓

* < ○ □

Затем мы выполняем команду синхронизации:

\$ синхронизация

После выполнения этой команды он откроет страницу администрирования в браузере нашего телефона, если она не откроется автоматически, мы можем перейти к любому браузеру, который мы установили, где мы обычно перемещаемся по Интернету, и мы можем поместить следующее !:

<http://127.0.0.1:8384>

Откроется экран администрирования инструмента, который поможет нам синхронизировать информацию между всеми узлами (телефонами) системы.



```
$ 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
[OEPS] 04:02:08 INFO: My ID: OWEPSNA-6RZI6S7-SK
VOU65-V44BC5Z-CXZOJI4-C3JC7HM-IPY4V35-JQAKPAW
[OEPS] 04:02:09 INFO: Single thread SHA256 perf
ormance is 12 MB/s using crypto/sha256 (12 MB/s
using minio/sha256-simd).
[OEPS] 04:02:11 INFO: Hashing performance is 11
.17 MB/s
[OEPS] 04:02:11 INFO: Migrating database to sch
ema version 1...
```



Так как у нас открыт экран администрирования "синхронизация", мы переходим к регистрации узла или узлов, которые мы хотим синхронизировать информацию. На данном этапе мы зайдем место программы, которая считывает QR-коды.

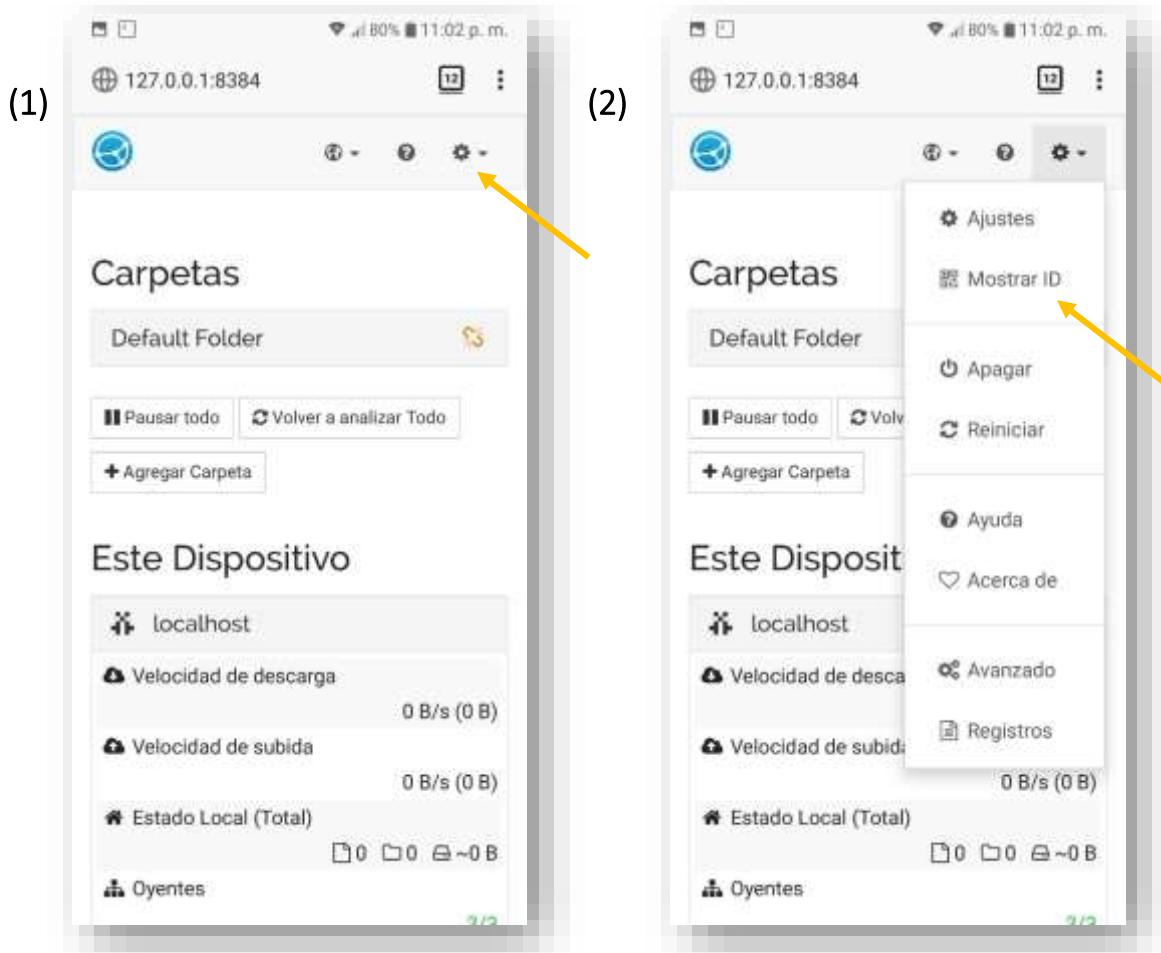
Программа синхронизации при ее первом запуске создает уникальный идентификатор телефона, который состоит из группы из восьми наборов буквенно-цифровых символов заглавными буквами, этот идентификатор (ID) является тем, который мы будем регистрировать в узле или узлах, которые мы хотим синхронизировать информацию.

В нашем случае телефонный номер LG Q6 должен быть зарегистрирован на телефон Sansumg, а телефонный номер Sansumg должен быть зарегистрирован на LG Q6. Они должны быть в обоих телефонах, чтобы работать правильно.

Мы выполним шаги по регистрации мобильного телефона Sansumg на телефоне LG Q6.

Сначала (1) в верхней части экрана администрирования (интернет-браузер) телефона Sansumg с синхронизацией мы перейдем на вкладку меню в верхней правой части.

На втором (2) шаге мы увидим меню, в котором нажимаем на "Показать ID" в Sansumg.

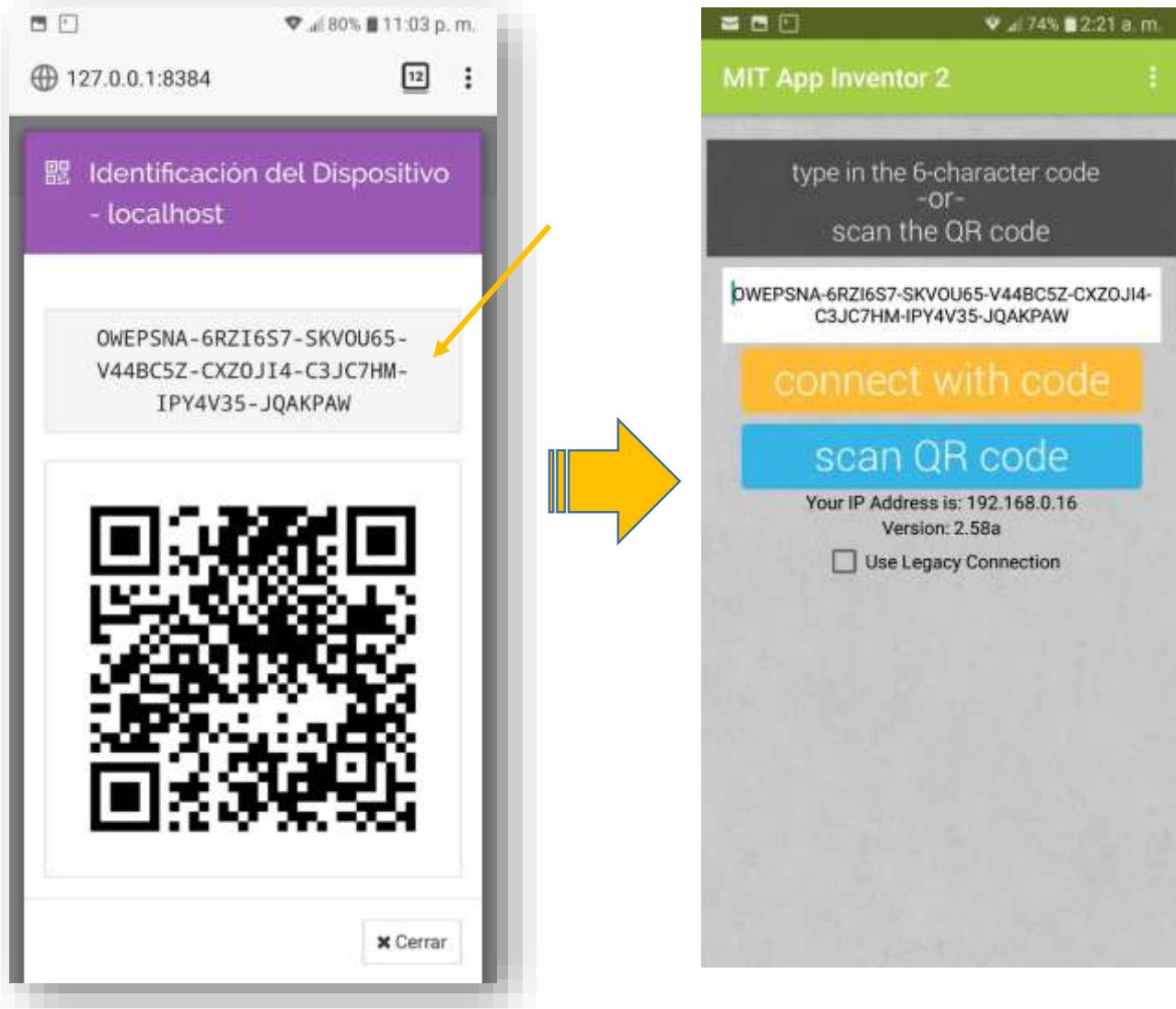


При нажатии на "Показать ID" появляется следующее окно, представляющее собой QR-код телефона Samsung, в нашем случае ID, идентифицирующий телефон - это

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

Тогда в телефоне LG Q6 программа, которая поможет нам запечатлеть этот QR-код Samsung, является той, которую мы предлагаем из приложения App Inventor (опционально), хотя в случае, если у нас его нет, мы также можем просто ввести его вручную, когда мы начнем регистрацию в LG Q6, однако, чтобы избежать любой ошибки, мы предлагаем использовать его.

Используя изобретателя программы App, установленной в мобильном телефоне LG Q6 для захвата QR-кода с удаленного телефона Samsung, мы хотим синхронизировать.



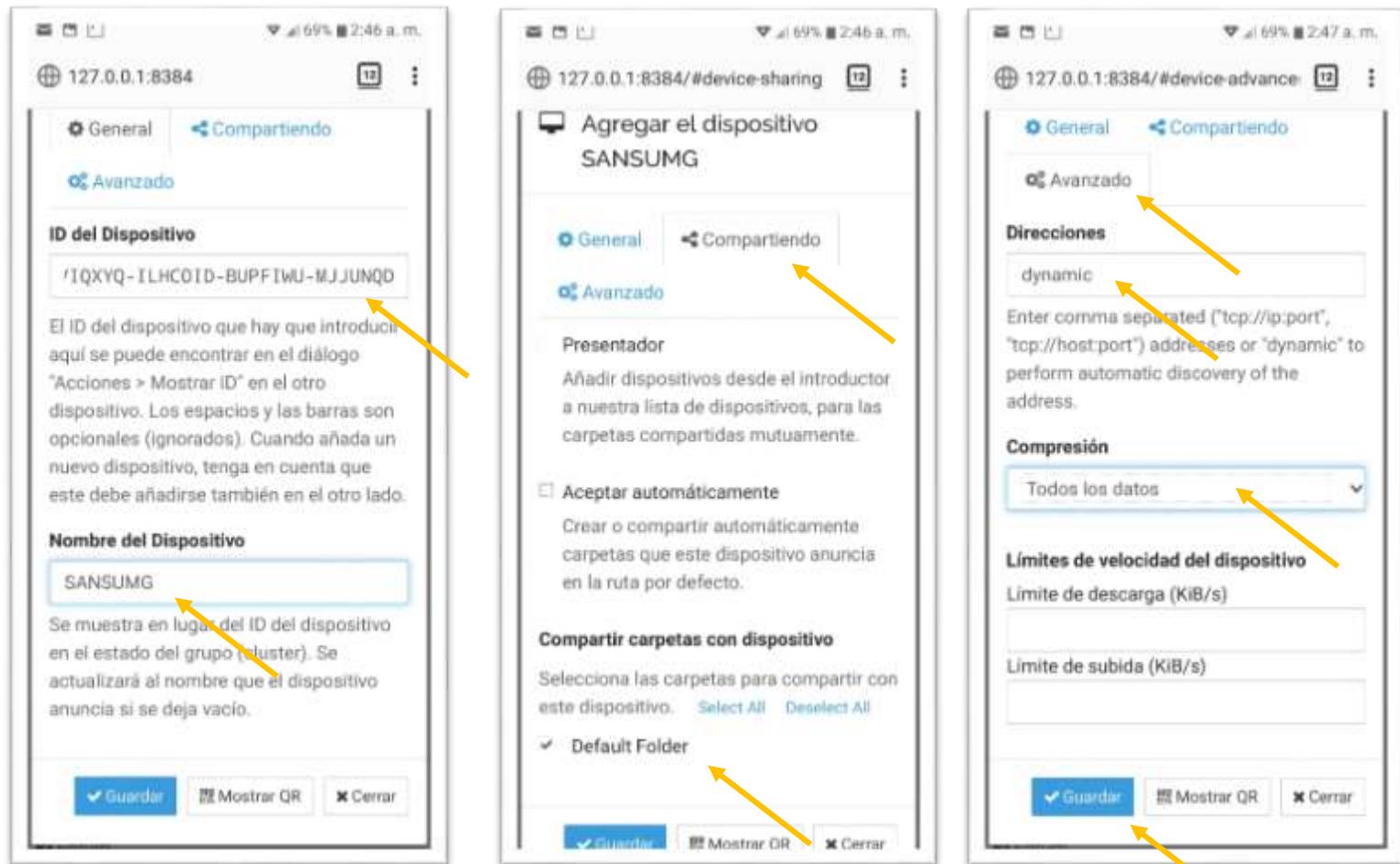
Затем мы копируем в буфер обмена QR-код в программе изобретателя приложений, нажав на захваченный код, выбираем "ВЫБРАТИТЬ ВСЕ" → "КОПИ".

С этой информацией мы приступаем к регистрации Сансумга в LG Q6. На экране администрирования мы переходим в нижнюю часть, где написано "Другие устройства" и нажимаем на кнопку "Добавить устройство", вводим идентификатор Samsung и присваиваем ему регистрационное имя.

Затем в верхней закладке "Share" мы нажимаем и в этом месте отмечаем нижнюю опцию в папке "Default", при этом мы будем совместно использовать каталог, который был создан по умолчанию под названием Sync в нашем устройстве.

Затем в верхней части мы переходим на вкладку "Дополнительно" и выбираем опцию сжатия данных в разделе "Все данные".

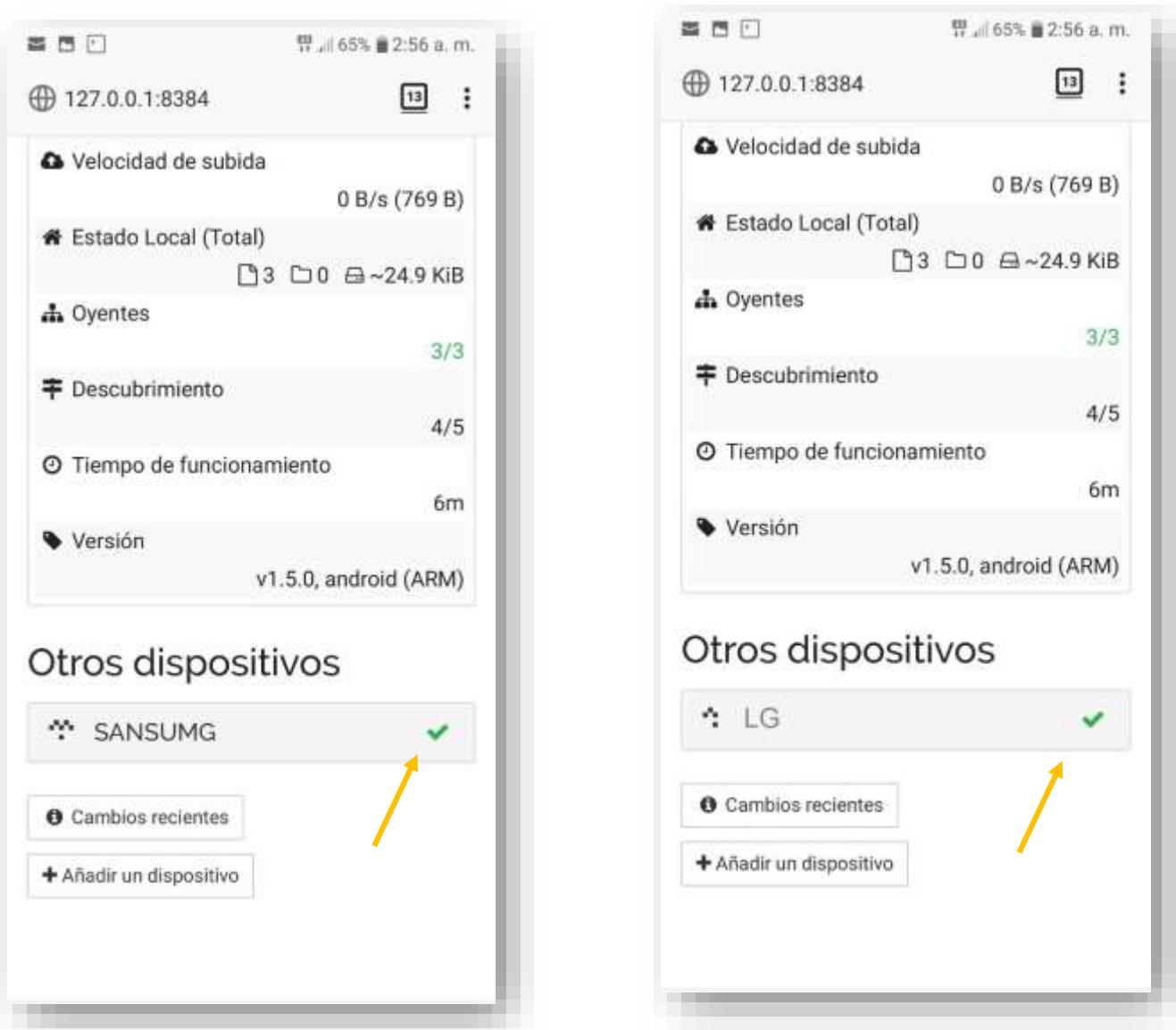
Наконец, мы нажимаем на нижнюю кнопку сохранения, мы завершаем регистрацию в узле, этот же процесс должен быть сделан в удаленном телефоне или телефонах, которые мы хотим синхронизировать.



При сохранении и регистрации в обоих телефонах мы будем ждать максимум 30 секунд, в течение которых находятся устройства, и соединение между устройствами будет выглядеть так же хорошо, как и при подтверждении зеленым цветом.

Устройство #1 LG Q6

Устройство № 2 Сансум



Вся информация в каталоге Sync, расположеннном по пути /data/data/com.termux/file/home/Sync будет синхронизирована, любые изменения будут скопированы и синхронизированы.

Пиринговая конфигурация системы с автоматической синхронизацией для использования в Mini BlocklyChain.

Теперь мы сделаем настройку в автоматическом режиме, ранее мы делали ручную настройку полезной, если мы работаем с минимальным количеством узлов, однако в случае наличия большого количества узлов это будет неэффективно, поэтому позже у нас будет инструмент SyncingManager, чтобы настроить его в автоматическом режиме с помощью автоматических онлайн-команд.

Пересмотреть конфигурацию базы данных для (ведомых) узлов.

Конфигурация файла `/data/data/com.termux/files/usr/etc/redis.conf` базы данных Redis (**Slave**) для мобильных телефонов Android.

Добавьте следующие изменения или директивы в файл, сохраните изменения и запустите сервер Redis.

Сначала найдите и удалите символ # из строки **slaveof**. Эта директива берет IP-адрес и порт, которые вы используете для безопасной связи с главным сервером Redis, разделенные пробелом. По умолчанию сервер Redis прослушивает на локальном интерфейсе 6379, но каждый из методов сетевой безопасности каким-то образом изменяет значение по умолчанию для других.

Значения, которые вы используете, будут зависеть от метода, который вы использовали для защиты вашего сетевого трафика:

Изолированная сеть: используйте IP-адрес изолированной сети и порт Redis (6379) ведущего сервера (например, slave простого IP_адреса 6379).

stunnel или **spiped**: используйте локальный интерфейс (127.0.0.1) и настроенный порт для туннелирования трафика.

PeerVPN: используйте IP VPN адрес главного сервера и обычный порт Redis.

Генерал изменит его:

ведомый ip_contact_server master_contact_port

Пример: **ведомый** 192.168.1.69 6379

masterauth your_network_master_password

Пример: **masterauth** sdfssdfsdfsd12WqE34Rfgthtdfd

requirepass your_network_slave_password

Пример: **requirepass** asdsjdsh34sds67sdFGbbnh

Сохраните и запустите сервер из терминала Termux следующей командой.

\$ redis-server redis.conf

После выполнения мы видим, как он синхронизируется с сервером Windows 10 (**Master**).

Файл конфигурации redis.conf \$ redis-server redis.conf

The image shows two side-by-side screenshots of a Termux terminal window. The left screenshot displays the command `$ redis-server redis.conf` being run, followed by a listing of files in the `/etc` directory. A yellow arrow points from the command to the file `redis.conf`. The right screenshot shows the Redis server logs during startup, indicating it is connecting to a master at 192.168.1.69:6379 and performing replication sync. A yellow arrow points from the logs to the message "Finished with success".

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

```
32672:5 31 May 2020 23:50:24.130 # Server initialized
32672:5 31 May 2020 23:50:24.131 * Loading RDB produced by version 6.0.1
32672:5 31 May 2020 23:50:24.131 * RDB age 27 seconds
32672:5 31 May 2020 23:50:24.131 * RDB memory usage when created 0.39 Mb
32672:5 31 May 2020 23:50:24.132 * DB loaded from disk: 0.001 seconds
32672:5 31 May 2020 23:50:24.132 * Ready to accept connections
32672:5 31 May 2020 23:50:24.132 * Connecting to MASTER 192.168.1.69:6379
32672:5 31 May 2020 23:50:24.136 * MASTER <-> REPLICAS sync started
32672:5 31 May 2020 23:50:24.159 * Non blocking connect for SYNC fired the event.
32672:5 31 May 2020 23:50:24.166 * Master replied to PING, replication can continue...
32672:5 31 May 2020 23:50:24.236 * Partial resynchronization not possible (no cached master)
32672:5 31 May 2020 23:50:24.266 * Full resync from master: 8ea52fe3c02ae241292f0dcbb823b8febcb09b784:0
32672:5 31 May 2020 23:50:24.349 * MASTER <-> REPLICAS sync: receiving 578 bytes from master to disk
32672:5 31 May 2020 23:50:24.353 * MASTER <-> REPLICAS sync: Flushing old data
32672:5 31 May 2020 23:50:24.353 * MASTER <-> REPLICAS sync: Loading DB in memory
32672:5 31 May 2020 23:50:24.354 * Loading RDB produced by version 5.0.7
32672:5 31 May 2020 23:50:24.354 * RDB age 0 seconds
32672:5 31 May 2020 23:50:24.354 * RDB memory usage when created 1.84 Mb
32672:5 31 May 2020 23:50:24.355 * MASTER <-> REPLICAS sync: Finished with success
```

Установка инструмента управления синхронизацией для узлов. Мы приступаем к установке **SyncthingManager**.

Сначала мы устанавливаем то, что вам нужно, чтобы SyncthingManager работал правильно.

\$ подходящая установка Python

\$ pip3 установка -дополнительный трубопровод

\$ npm установка синхронизатора

Инструмент SyncThingManager поможет нам синхронизировать "peer to peer" автоматически, а не вручную для новых и существующих узлов.

The image consists of three side-by-side screenshots of a terminal window on a mobile device. Each screenshot shows a command being entered at the prompt '\$'. A yellow arrow points to each command. The background of the terminal window is black, and the text is white. The status bar at the top of each screenshot shows signal strength, battery level (92%), and time (e.g., 2:48 a.m.).

- Screenshot 1:** Shows the command \$ apt install python. The output indicates that Python is already the newest version (3.8.3) and nothing is upgraded.
- Screenshot 2:** Shows the command \$ pip3 install --upgrade pip. The output shows that pip is already up-to-date at version 20.1.1.
- Screenshot 3:** Shows the command \$ pip3 install syncthingmanager. The output lists various dependencies being installed, including syncthing, python-dateutil, six, chardet, idna, certifi, and urllib3.

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

App Inventor - это среда разработки программного обеспечения, созданная Google Labs для создания приложений для операционной системы Android. Пользователь может, визуально и из набора основных инструментов, связать ряд блоков для создания приложения. Система бесплатна и может быть легко загружена из Интернета. Приложения, созданные с помощью App Inventor, очень легко создавать, так как не требуется знание какого-либо языка программирования.

Все текущие среды, использующие технологию Blockly, такие как AppyBuilder и Thunkable, среди прочих, имеют свою бесплатную версию, их способ использования может быть через интернет на различных сайтах, или они также могут быть установлены дома.

Блоки, входящие в архитектуру Mini BloclyChain, были протестированы в App Inventor и AppyBuilder, но из-за оптимизации кода они должны работать на других платформах.

Онлайн-версии:

App Inventor.

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

AppyBuilder.

<http://appybuilder.com/>

Спасибо.

<https://thunkable.com/>

Версия, которая будет установлена на вашем компьютере (ПК):

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

Среда для разработчиков блочных блоков.

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

18. Что такое Доказательство Кванта (PQu)?

Поку. - "Доказательство Кванта" - это алгоритм консенсуса, разработанный для Mini BlocklyChain, данный тест является вариантом теста работы (PoW), который работает следующим образом.

Тест кванта (PoQu) при запуске выполняется по тому же алгоритму, что и "Тест работы" (PoW), основанный на постановке процессора устройства (ПК, сервера, планшета или мобильного телефона) на работу для получения строки символов, представляющей собой математическую головоломку, называемую "хэш".

Помните, что "хэш" - это алгоритм или математический процесс, который при введении фразы или какого-то вида цифровой информации, такой как текстовые файлы, программа, изображение, видео, звук или другой разнородный вид цифровой информации, дает нам в результате буквенно-цифровой символ, представляющий собой цифровую подпись, которая представляет собой уникальную и неповторяющую форму данных, алгоритм хэширования является односторонним, это означает, что при вводе данных для получения их "хэша" подписи, обратный процесс не может быть выполнен, имея "хэш" подписи, мы не можем знать, какая информация была получена этим свойством, что дает нам преимущество в безопасности при обработке информации, которую мы посылаем через Интернет. Как это работает? Представьте себе, что посылая любую информацию по небезопасным каналам и сопровождая ее соответствующим "исходным хэшем", получатель при получении информации может получить "хэш" полученной информации, мы назовем его "целевой хэш" и проверим его "исходным хэшем", если оба "хэша" одинаковы, мы можем подтвердить, что информация не была изменена в канале, который был отправлен, это всего лишь пример, где этот тип процесса информационной безопасности используется в настоящее время.

В настоящее время существуют различные типы алгоритмов или хэш-процессов, которые отличаются уровнем безопасности. Наиболее используемые или известные: MD5, SHA256 и SHA512.

Пример SHA256:

У нас есть следующая цепочка или предложение: "Мини БлоклиЦеня" модульная.

Если мы применим хэш SHA256 к предыдущей строке, то получим следующий хэш.

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9of97cb2afd827e8804db8

Вышеуказанная буквенно-цифровая строка является подписью, которая представляет предложение в вышеприведенном примере

Для более подробного примера можно воспользоваться сайтом в интернете:

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

В случае алгоритма "Test Work" (PoW), он работает, используя вычислительную мощность, для получения предопределенного хэша.

Давайте представим, что у нас есть предыдущий "хэш", который мы взяли из цепи "Mini BlocklyChain is modular".

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9of97cb2af827e8804db8

К этому "хэшу" в его начале мы добавляем параметр сложности, то есть просто ставим нули "0" в начале, то есть если мы скажем, что сложность равна 4, то он будет иметь "**0000**" + "хэш", то мы назовем его "семенной хэш".

0000 f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db8

Теперь, учитывая, что мы знаем входную информацию, которая является строкой: "Mini BlocklyChain является модульной", мы добавляем в конце строки число, начинающееся с нуля "0", и вынимаем его хэш к этому мы будем называть его "хэш нонсе":

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db80

У нас есть гашиш нонс:

7529f3ad273fc8a9eff12183f8d6f886821900750bb6b59c1504924dfd85a7c8

Затем выполняется сравнение нового "хэша nonce" с "семенем хэша", если они равны узлу, который первым найдет равенство, то выиграет выполнение обработки текущей транзакции. Как видим, этот процесс основан на вероятности и вычислительной силе устройства, которое дает тест "Доказательство работоспособности" равенство консенсуса для всех узлов.

Если "семенной хэш" не совпадает с "хэшем nonce", сложность увеличивается на единицу и "хэш nonce" снова удаляется, увеличивающее число называется "nonce", оно сравнивается с "семенным хэшем" до тех пор, пока они не совпадут или не станут одинаковыми.

Как видим, число "nonce" или увеличение - это число, которое поможет получить "хэш" равенства.

Основанный на алгоритме "Test of Work" (PoW), алгоритм "Test of Quantum" (PoQu) основан на получении числа "nonce", как и PoW, и при использовании минимального уровня сложности от 1 до 5, это служит только мобильному устройству для получения права быть кандидатом на победу в конкурсе.

Квантовый тест (PoQu), активируется, когда мобильный телефон заканчивает минимальный PoW и выигрывает проход для получения вероятностного номера в системе QRNG.

QRNG (Quantum Random Number Generator - Генератор Квантовых Случайных чисел) - это Генератор Квантовых Случайных чисел, эта система основана на генерации истинных случайных чисел на основе квантовой механики и на сегодняшний день является самой безопасной системой для генерации таких чисел. Подробнее см. приложение "Квантовые вычисления с OpenQbit".

Mini BlocklyChain может реализовать как минимальный тип концессии PoW, так и тип концессии PoQu.

Тест PoQu основан на получении числа "nonce", это число в тесте PoQu известно как "Магическое число", при этом система "Peer to Peer" будет подтверждать правильность числа, а затем случайное число будет получено с пулом серверов QRNG. Это случайное число будет зарегистрировано во всех узлах, будет создан список, содержащий **((Node Sum /2)) +1** и из этого списка будет выбран тот, который с наибольшей вероятностью станет кандидатом-победителем консенсуса (PoQu), и этот список выполнит текущую очередь транзакций.

Алгоритм PoQu также использует тестирование **NIST** (Национального института стандартов и технологий), чтобы убедиться, что случайные числа в QRNG действительно являются случайными числами.

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

В Mini BlocklyChain мы реализовали блок для PoW и блок для PoQu.

Эти блоки используют тип хэша: SHA256 для свободного использования, для коммерческого использования у вас есть SHA512 и другие типы хэшей по мере необходимости.

Более подробная информация о концепции HASH приведена ниже:

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

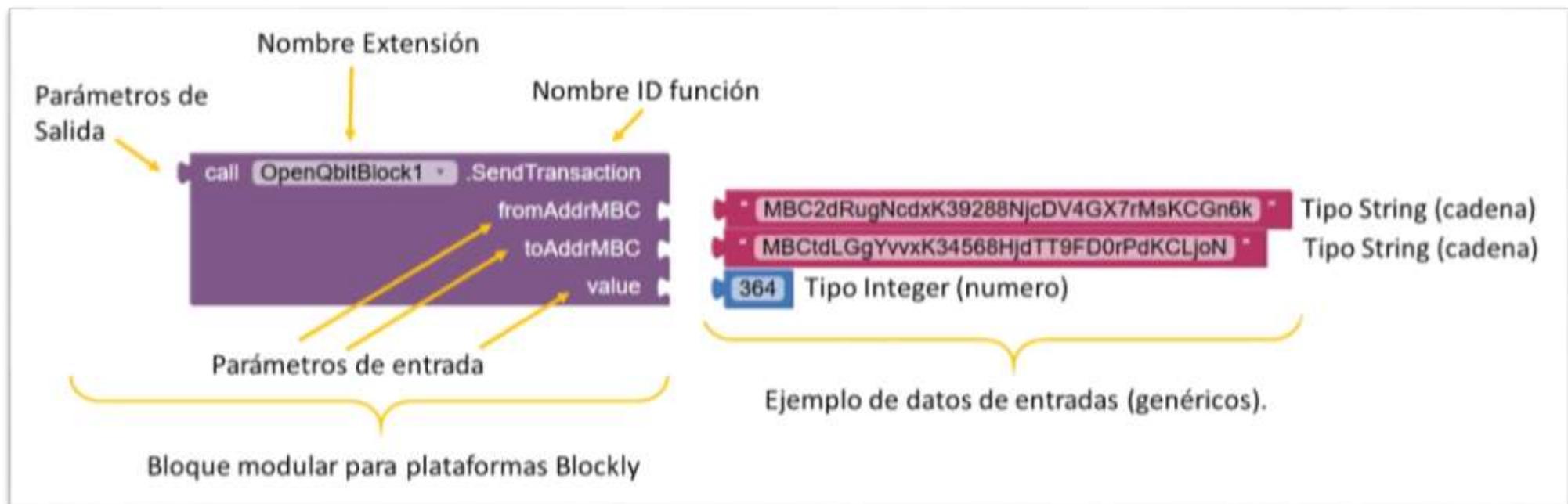
ПРИМЕЧАНИЕ: Тест работы (PoW), используемый в мобильных телефонах, может использовать только максимальную трудность 5, так как математическая обработка этих устройств не выделена, как серверы или ПК. Мы используем алгоритм PoW только для того, чтобы получить Ваш пропуск или разрешение на вход в систему Генератора Квантовых Случайных Номеров (QRNG) и с его помощью выполнить алгоритм Генератора Квантовых Случайных Номеров (PoQu).

На мобильных телефонах не используйте максимальную сложность 5, так как система может заблокироваться и не реагировать должным образом.

19. Определение и использование блоков в Mini BlocklyChain

Мы начнем с объяснения распределения данных, которые будут иметь все блоки, их синтаксис использования и настройки.

В следующем примере мы видим модульный блок и его входные и выходные параметры, а также типы входных данных, эти данные могут быть типа String (строка символов) или Integer (целое или десятичное число). Мы показываем, как он используется и настраиваем его для правильного функционирования.



Каждый блок модуля будет иметь свое описание и будет назван в случае, если он имеет обязательные или необязательные зависимости от других блоков, используемых в качестве входных параметров, будет объявлен процесс интеграции.

Блок для создания временного списка строк в предопределенном массиве, называемом внутренне "цепочкой" - (**AddHash**).

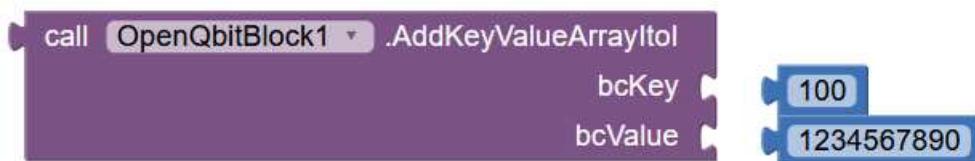


Входные параметры: **блок <стринг>**

Выходные параметры: Не применимо.

Описание: Блок для хранения временного массива хэшей или строк в предопределенном массиве, называемом внутренне "цепочкой".

Блок для создания массивов с ключевыми значениями (**Целочисленный Интегратор**) - (**AddKeyValueArrayItol**)



Входные параметры: **bcKey <Integer>, bcValue <Integer>**.

Выходные параметры: Возвращает значения, введенные на входе.

Описание: Это временное расположение значений ключей, оно предопределено внутренним именем "**Itol_UTXO**", это полезно для обработки UTXO-транзакций.

Блок для создания массивов с ключевыми значениями (**Integer-String**) - (**AddKeyValueArrayItos**)

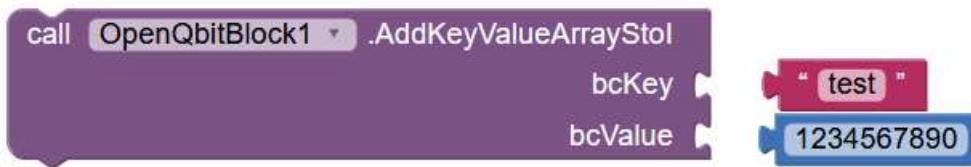


Входные параметры: **bcKey <Integer>, bcValue < String>**.

Выходные параметры: Возвращает значения, введенные на входе.

Описание: Это временное расположение значений ключей, оно предопределено внутренним именем "**Itos_UTXO**", это полезно для обработки UTXO-транзакций.

Блок для создания массивов значений ключей (**String-Integer**) - (**AddKeyValueArrayItoI**)



Входные параметры: **bcKey** <String>, **bcValue** <Integer>.

Выходные параметры: Возвращает значения, введенные на входе.

Описание: Это временное расположение клавиш, оно предопределено внутренним именем "**StoI_UTXO**", что полезно для обработки UTXO-транзакций.

Блок для создания массивов с ключевыми значениями (**String-String**) - (**AddKeyValueArrayStoS**)

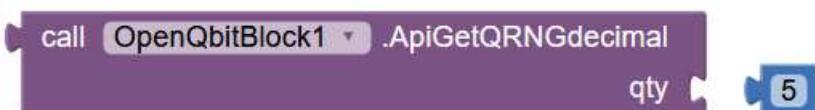


Входные параметры: **bcKey** <String>, **bcValue** <String>.

Выходные параметры: Возвращает значения, введенные на входе.

Описание: Это временное расположение клавиш, оно предопределено внутренним именем "**StoS_UTXO**", что полезно для обработки UTXO-транзакций.

Блок для генерации десятичных случайных квантовых чисел - (**ApiGetQRNGdecimal**)



Входные параметры: **qty** < Целый >

Выходные параметры: Дает количество "qty" случайных квантовых десятичных чисел, введенных во входные числа в диапазоне 0 и 1 в формате JSON.

Пример:

qty = 5; вывод: {"результат": [0.5843012986202495, 0.7746497687824652, 0.05951126805960929, 0.1986079055812694, 0.03689783439899279] }

Описание: API генератора квантовых случайных чисел (QRNG)

Блок для генерации десятичных случайных квантовых чисел - (**ApiGetQRNGdecimal**)



Входные параметры: **qty <Integer>**, **min <Integer>**, **max <max>**.

Выходные параметры: Дает количество "qty" случайных квантовых чисел, введенных во входные данные, числа находятся в диапазоне min и max в формате JSON.

Пример:

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

Описание: API генератора квантовых случайных чисел (QRNG)

Хэш-блок "SHA256" для строк символов (**ApplySha256**).



Входные параметры: **вход <String>**

Выходные параметры: вычисляет хэш "SHA256" символьной строки.

Пример:

Вход = "Мини-блочная цепь модульная".

выход: f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db8

Описание: Функция удаления хэша "SHA256". *Ша* или *SHA* относятся к: Алгоритм безопасного хэша - набор хэш-функций, разработанный Агентством национальной безопасности США. SHA256 использует 256-битный алгоритм.

Блок для очистки предопределенного внутреннего массива "цепочки" (**ClearBlockList**).

call OpenQbitBlock1 .ClearBlockList

Входные и выходные параметры: Не применимо

Описание: Удалить все элементы, имеющие внутреннее временное расположение "цепочки".

Блок для очистки предопределенного внутреннего массива (**Integer-Integer**) - (**ClearItoi**).

call OpenQbitBlock1 .ClearItoi

Входные и выходные параметры: Не применимо

Описание: Удалить все элементы, имеющие внутреннее временное расположение "Itoi_UTXO".

Блок для очистки предопределенного внутреннего массива (**Integer-String**) - (**ClearItoS**).

call OpenQbitBlock1 .ClearItoS

Входные и выходные параметры: Не применимо

Описание: Удалить все элементы, имеющие внутреннее временное расположение "ItoS_UTXO".

Блок для очистки предопределенного внутреннего массива (**String-Integer**) - (**ClearStoI**).

call OpenQbitBlock1 .ClearStoI

Входные и выходные параметры: Не применимо

Описание: Удалить все элементы, имеющие внутреннее временное расположение "StoI_UTXO".

Блок для очистки предопределенного внутреннего массива (**String-String**) - (**ClearStoS**).

call OpenQbitBlock1 .ClearStoS

Входные и выходные параметры: Не применимо

Описание: Удалить все элементы, имеющие внутреннее временное расположение "StoS_UTXO".

Блок для декодирования строки в Base10. (**DecodeBase58**)



Входные параметры: **input<String>**

Выходные параметры: Он передает оригинальную строку, которая была использована в блоке (**EncodeBase58**).

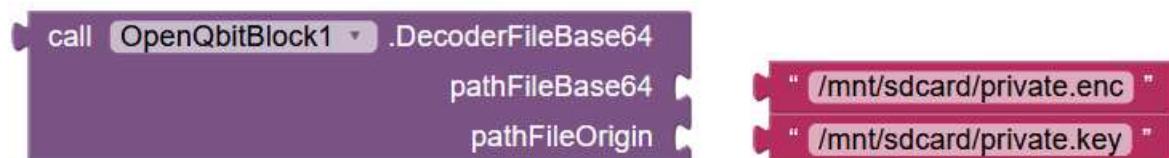
Пример:

Вход = oBRN8Aj67andJsbRGHfNSF9PTdZYGandVWrwMW7mTX

Выход: "Mini BlocklyChain - модульный".

Описание: Преобразует строку Base58 в исходный текст, заданный в блоке (**EncodeBase58**).

Блок декодирования файла с алгоритмом Base64 (**DecoderFileBase64**).



Входные параметры: **pathFileBase64 <String>**, **pathFileOrigin <String>**.

Выходные параметры: Исходный файл, который был введен в блоке (**EncoderFileBase64**)

Описание: Файл Base64 преобразуется в исходный файл, который был вставлен в блок (**EncoderFileBase64**).

Блок, чтобы узнать, пуст ли предопределенный внутренний массив "цепочки". (**EmptyBlockList**)



Входные параметры: Не применимо.

Выходные параметры: возвращает "True", если данные пустые, или "False", если данные есть.

Описание: Блок, спрашивающий, есть ли элементы предопределенного временного внутреннего массива "chain".

Блок для кодирования строки символов в Base58. (**EncodeBase58**).



Входные параметры: **input<String>**

Выходные параметры: Он передает оригинальную строку, которая была использована в блоке (**EncodeBase58**).

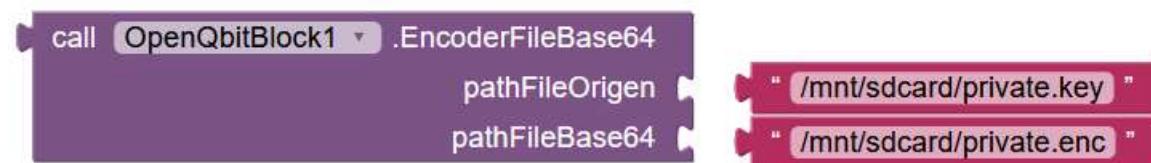
Пример:

Вход = "Мини-блочная цепь модульная".

Наша производительность: oBRN8Aj67yJsbRGHfNSF9PTdZYGyVWrwMW7mTX

Описание: Преобразует нагрудную цепь в цепь на Base58. Алгоритм Base58 представляет собой группу бинарно-текстовых схем кодирования, используемых для представления больших целых чисел в виде буквенно-цифрового текста, введённую Сатоши Накамото для использования с Bitcoin.

Блок для кодирования файла с алгоритмом Base64 (**EncoderFileBase64**).

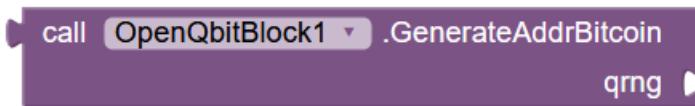


Входные параметры: **pathFileOrigin <String>**, **pathFileBase64 <String>**.

Выходные параметры: файл в кодировке Base64.

Описание: Преобразует исходный файл любого формата в файл Base64. Имена файлов могут быть произвольными и выбираться пользователем.

Блок Генератор адресов пользователей (**GenerateAddrBitcoin**).



Обязательная единица: Блок (**ApiGetQRNGinteger**),

Зависимости (опционально): расширение **OpenQbitFileEncription**, **OpenQbitFileDecryption** и **OpenQbitSQLite**.

Входные параметры: **qrng < обязательная зависимость>**

Выходные параметры: 34-символьный буквенно-цифровой адрес транзакции и адрес использования keyStore

Описание: Блок для создания нового общего адреса транзакции Bitcoin для пользователя и генератора приватных ключей (Цифровая подпись для отправки транзакций) и публичного ключа (Публичный адрес для совершения транзакций). Этот генератор ключей в основном является генератором адресов для использования в цифровом кошельке или обычно называется "кошелек". Этот блок используется совместно с блоками Генератора квантовых случайных чисел (QRNG), и два выходных параметра должны быть вставлены в KeyStore.

KeyStore - это база данных, в которой хранятся приватные ключи в шестнадцатеричном формате:

Пример шестнадцатеричного адреса, который хранится в KeyStore

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

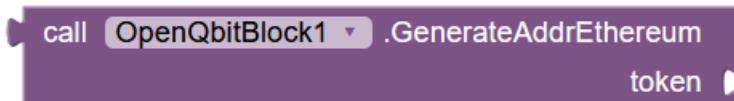
Эта база используется только локальным пользователем и информация шифруется, этот процесс осуществляется с помощью расширения **OpenQbitSQLite** и **OpenQbitAESEncryption** и **OpenQbitAESDecryption** расширений шифрования информации.

См. приложение "Создание хранилища ключей". Генерируемые адреса используют тот же алгоритм адресации Bitcoin, при этом начальный идентификатор адреса bitcoin равняется "1".

Адреса, генерируемые блоком (**GenerateAddrBitcoin**), составляют 34 буквенно-цифровых символа, состоящих из 33 буквенно-цифровых символов и 1 идентификатора Bitcoin следующим образом:

12dRugNcdxK39288NjcDV4GX7rMsKCGn6k

Блок генератора адресов пользователей (`GenerateAddrEthereum`).



Принудительная зависимость: получите жетон по адресу:
<https://accounts.blockcypher.com/signup>.

Зависимости (опционально): расширение `OpenQbitFileEncription`, `OpenQbitFileDecryption` и `OpenQbitSQLite`.

Входные параметры: **токен** < обязательная зависимость - строка>.

Выходные параметры: 40 буквенно-цифровых символов адреса транзакции не включают в себя начальный Ethereum индикатор "0x", который дал бы нам 42 символа общего адреса. Он также возвращает открытый и закрытый ключи.

Пример:

```
{  
    "частный":  
        "227ac59f480131272003c2d723a77795ebd3580acaab62b5c537989e2ce4e08ef",  
    "общественный":  
        "04e2d55ebccd32a7384e096df559cc36b856c64a16e5b402e10585dc3ea055672aafa84df8  
        a859531570a650a8ab1e7a22949100efa1a5f072c035551cac1ce",  
    "адрес": "14e150399b0399f787b4d6fe30d8b251375f0d66"}
```

Описание: Блок для создания нового адреса транзакции для пользователя и генератора частного ключа (Цифровая подпись для отправки транзакций) и публичного ключа (Публичный адрес для совершения транзакций). Этот генератор ключей является внешним API и вы должны иметь Wifi или мобильное соединение, это в основном генератор адресов, чтобы использовать его в цифровом кошельке или обычно называют "кошелек".

Вы можете создать aKeyStore - базу данных, в которой хранятся приватные ключи в шестнадцатеричном формате, см. приложение "Создание хранилища ключей".

Пример шестнадцатеричного адреса, который хранится в KeyStore

0x14e150399b0399f787b4d6fe30d8b251375f0d666

Закрытый ключ используется только локальным пользователем, а информация шифруется, этот процесс осуществляется с помощью расширений OpenQbitSQLite и OpenQbitAESEncryption и OpenQbitAESDecryption.

Блок генератора адресов пользователей (`GenerateAddrMiniBlocklyChain`).



Обязательная единица: Блок (`ApiGetQRNGinteger`),

Зависимости (опционально): расширение `OpenQbitFileEncription`, `OpenQbitFileDecryption` и `OpenQbitSQLite`.

Входные параметры: `qrng < обязательная зависимость>`

Выходные параметры: 36 буквенно-цифровой адрес транзакции и адрес использования keyStore. Метод блочного обзора (`PairKeysMBC`).

Описание: Блок для создания нового адреса транзакции для пользователя и генератора частного ключа (Цифровая подпись для отправки транзакций) и публичного ключа (Публичный адрес для совершения транзакций). Этот генератор ключей в основном является генератором адресов для использования его в цифровом кошельке или обычно называется "кошелек".

Этот блок используется совместно с блоками Генератора квантовых случайных чисел (QRNG), и два выходных параметра должны быть вставлены в KeyStore.

KeyStore - это база данных, в которой хранятся приватные ключи в шестнадцатеричном формате:

Пример шестнадцатеричного адреса, который хранится в KeyStore

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

Эта база используется только локальным пользователем и информация шифруется, этот процесс осуществляется с помощью расширения `OpenQbitSQLite` и `OpenQbitAESEncryption` и `OpenQbitAESDecryption` расширений шифрования информации.

См. приложение "Создание хранилища ключей". Генерируемые адреса используют один и тот же алгоритм адреса bitcoin, единственное отличие состоит в том, что идентификатор адреса bitcoin, который равен "1" или "3", меняется на Mini BlocklyChain идентификатор "MBC".

Адреса, генерируемые блоком (**GenerateAddrMiniBlocklyChain**), состоят из 36 буквенно-цифровых символов и 3 заглавных букв "МВС" мини-идентификатора BlocklyChain:

Mbc2dRugNcdxK39288NjcDV4GX7rMsKCGn6k

Блочный метод (**PairKeysMBC**) - это блочный выход (**GenerateAddrMiniBlocklyChain**).



Входные параметры: Не применимо.

Выходные параметры: **addressMBC <String>**, **privateKeyHex <String>**.

Описание: Доставлены публичный адрес пользователя в формате Mini BlocklyChain и приватный ключ в шестнадцатеричном формате.

Пример:

адресMBC: MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k

privateKeyHex:

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

Блок Генератор адресов пользователей (**GenerateKeyValuePair**).



Обязательная единица: Блок (**ApiGetQRNGinteger**),

Зависимости (опционально): расширение **OpenQbitFileEncryption**,
OpenQbitFileDecryption и **OpenQbitSQLite**.

Входные параметры: **qrng < обязательная зависимость>**

Выходные параметры: Предоставляет публичный и первичный ключ для локального пользователя

Описание: Генерирует открытый и первичный ключи, используя алгоритм ECC (1) и шестнадцатеричный формат.

- (1) Криптография эллиптических кривых (ECC) - это вариант асимметричной криптографии или криптографии с открытым ключом, основанный на математике эллиптических кривых.

Блокировка для подписания транзакций (**GenerateSignature**).

call OpenQbitBlock1 .GenerateSignature

Необходимая зависимость: Блок (**GenerateKeyPairs**), Блок (**SenderLoadKeyPair**), Блок (**RecipientLoadKeyPair**). Представьте эти блоки перед использованием.

Входные параметры: <Обязательные контрольные зависимости>

Выходные параметры: Нет выхода.

Описание: Генерирует цифровую подпись Отправителя, применяемую к активу, отправленному в сделке Получателю, эта подпись будет подтверждена, когда сделка обрабатывается каким-либо узлом сети, с этой подписью система Mini BlocklyChain гарантирует, что актив не был изменен в отправленном и что он соответствует отношениям Отправитель-получатель и не перенаправлен и не применен на другой цифровой адрес.

Блок для получения общего баланса активов пользователя (**GetBalance**).

call OpenQbitBlock1 .GetBalance

fromAddrMBC

" MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k "

Обязательная зависимость: Блок (**ConnectorTransactionTail**).

Параметры ввода: изTransTail <Строка массива>, <Принудительные зависимости платы>.

Выходные параметры: Проверяет расходы на входящие и исходящие активы по каждой транзакции.

Описание: Проверяет баланс для утверждения, если у пользователя есть активы для отправки или если активы, которые он получает, добавляются к его балансу.

Блок для получения q22 с мобильного телефона. (**GetDeviceID**)

call OpenQbitBlock1 .GetDeviceID

Входные параметры: **Не применимо**

Исходящие параметры: передает внутренний идентификатор мобильного телефона.

Описание: Предоставляет уникальный для каждого устройства идентификатор мобильного телефона IMEI.

Блок для удаления хэша RIPEMD-160 из строки. (**Ripemd160**)

call OpenQbitBlock1 .GetHashRipemd160
str " Mini BlocklyChain es modular "

Входные параметры: **str <String>**

Выходные параметры: вычисляет хэш символьной строки "RIPEMD-160".

Пример:

Вход = "Мини-блочная цепь модульная".

выход: ae29436e4b8ea8ed6143f3f92380dfa2f4f47336

Описание: Получить хэш "RIPEMD-160". Этот хэш используется в процессе генерации действительного адреса для Bitcoin и Mini BlocklyChain.

RIPEMD-160 (аббревиатура от RACE Integrity Primitives Evaluation Message Digest) - это 160-битный алгоритм дайджеста сообщений (и криптографическая хеш-функция).

Блок для расчета дерева Мерклера. (**GetMerkleRoot**)

call OpenQbitBlock1 .GetMerkleRoot
transactions make a list
" MBCoBRN8Aj67yJsbRGHfNSF9PT:2349 "
" MBCdZYGVW246tRFgrwMW7mTX:8900 "
" MBChGDFTREBcSSdfgssBNMcVc:2356 "
" MBCJu34R5TY67fSSdfgssONERFc:1033 "
" MBC....(N-1) "
" MBC.....N "

Входные параметры: транзакции <Строка массива>

Выходные параметры: выдает хэш-тип "SHA256".

Пример:

Input = очередь транзакций, расположение всех обрабатываемых транзакций.

выход: b4a44c42b6070825f763cd118d6ab49a8e80bbb7cdc0225064f8e042b94196bd

Описание: Получить хэш "SHA256", используя алгоритм дерева Меркл.

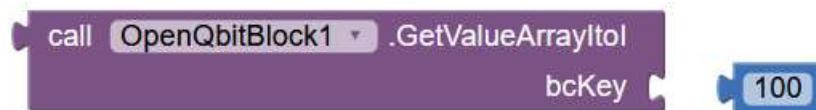
Merkle Hash Tree - это бинарная или небинарная структура дерева данных, в которой каждый узел, не являющийся листом, помечен хэшем конкатенированных меток или значений (для узлов листа) его дочерних узлов. Это обобщение хэш-списков и хэш-строк.

Это позволяет связать большое количество отдельных данных с одним значением хэша - хэшем корневого узла дерева. Это обеспечивает безопасный и эффективный метод проверки содержимого больших структур данных. В его практическом применении хэш корневого узла обычно подписывается, чтобы гарантировать его целостность и абсолютную надежность проверки. Демонстрация того, что узел листа является частью заданного хэш-дерева, требует количества данных, пропорционального логарифму количества узлов в дереве.

В настоящее время основное применение деревьев Merkle заключается в том, чтобы сделать безопасными блоки данных, полученные от других коллег в одноранговых сетях, чтобы обеспечить их получение без повреждений и без изменений.

Он используется для проверки того, что очередь, в которой находятся обрабатываемые транзакции, не была изменена, а также для обеспечения ее целостности для рассеивания во всех узлах сети Mini BlocklyChain. Все узлы должны выполнять этот алгоритм для обеспечения целостности каждой транзакции, которая будет применяться.

Блок для получения значения из предопределенного массива "Itol_UTXO" (GetValueArrayItol).

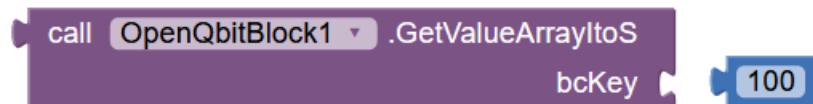


Входные параметры: **bcKey** <Integer>

Выходные параметры: Возвращает значение <Integer>, сохраненное в метке с заданным номером в качестве входного.

Описание: Речь идет о запросе на временное расположение значений ключей, которое предопределено внутренним именем "ItoL_UTXO", что полезно для обработки UTXO-транзакций.

Блок для получения значения из предопределенного массива "ItoS_UTXO" (**GetValueArrayItoS**).

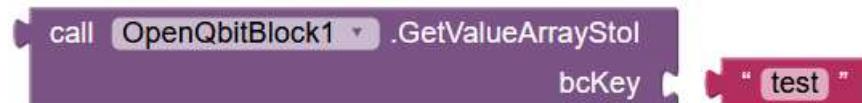


Входные параметры: **bcKey** <Integer>

Выходные параметры: Возвращает значение <String>, сохраненное в метке с заданным номером в качестве входного.

Описание: Речь идет о запросе на временное расположение значений ключей, которое предопределено внутренним именем "ItoS_UTXO", что полезно для обработки UTXO-транзакций.

Блок для получения значения из предопределенного массива "StoL_UTXO" (**GetValueArrayStoL**).

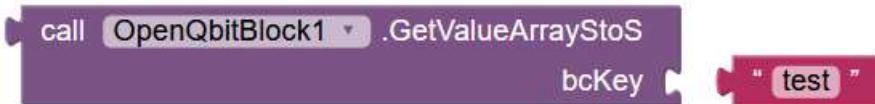


Входные параметры: **bcKey** <Строка>

Выходные параметры: Возвращает значение <Integer>, сохраненное в метке с заданным именем в качестве входного.

Описание: Речь идет о запросе на временное расположение значений ключей, которое предопределено внутренним именем "StoL_UTXO", что полезно для обработки UTXO-транзакций.

Блок для получения значения из предопределенного массива "StoS_UTXO" (**GetValueArrayStoS**).



Входные параметры: **bcKey** <Строка>

Выходные параметры: Возвращает значение <String>, сохраненное в метке с заданным именем в качестве входного.

Описание: Речь идет о запросе на временное расположение значений ключей, которое предопределено внутренним именем "StoS_UTXO", что полезно для обработки UTXO-транзакций.

Блочный валидатор цепи блока. (**IsChainValid**)

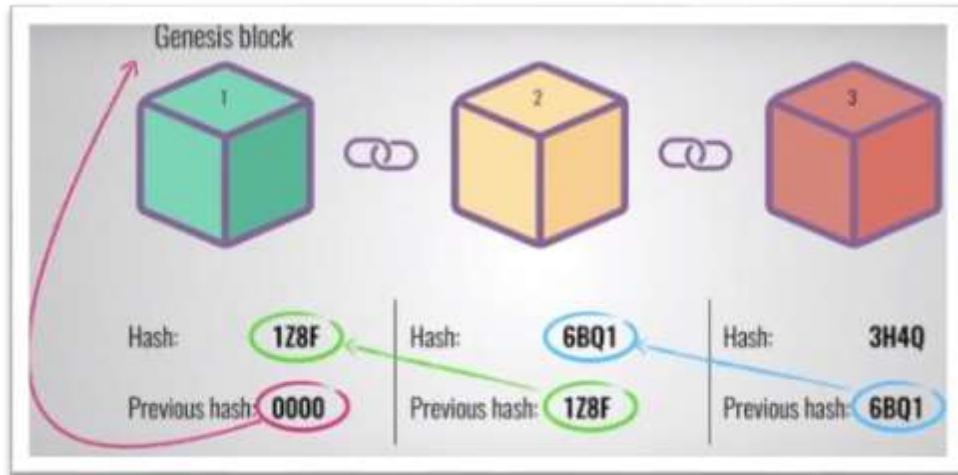


Необходимая зависимость: Блок (**GenerateKeyPairs**), Блок (**SenderLoadKeyPair**), Блок (**RecipientLoadKeyPair**), Блок (**GenerateSignature**). Представьте эти блоки перед использованием.

Входные параметры: Не применимо.

Выходные параметры: Доставка "True", если валидация строки блока правильная или "False", если валидация не удалась.

Описание: Он обеспечивает нам валидацию компонентов, которые были ранее вставлены в систему блок-цепочки, эта валидация является наиболее важной из всей системы в целом. Как работает проверка цепочки блоков или как она известна в общем виде (BlockChain). Это основная часть каждой системы.



Как видно из предыдущего образа, каждый блок, который добавляется в систему хранения, связан с предыдущим блоком через хэш-алгоритмы.

Например, при создании нового блока для добавления, хэш, представляющий новую информацию, получается путем взятия хэша нового информационного блока + предыдущий хэш. При этом типе звена будет обнаружено любое минимальное изменение в хранении блочных цепочек, что позволяет обеспечить очень высокую и эффективную защиту данных.

Ниже приведен пример того, как строка блоков хранится в БД SQLite, мы увидим, как предыдущий хэш связан с новым хэшем последнего вставленного блока (последней очереди обработанных транзакций). Каждый хэш в колонках "prevhash" и "newhash" представляет собой информацию об очередности транзакций, которая была обработана в тот момент.

The screenshot shows the SQLite Expert Personal 5.3 interface. The database is named 'miniblock' and the table is 'nblock'. The table structure includes columns: rowid, id, prevhash, newhash, opera, trans, and balan. The data grid displays three rows of data:

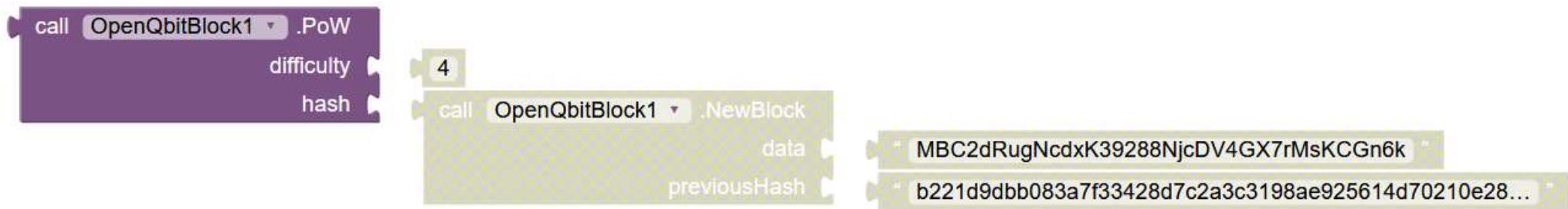
rowid	id	prevhash	newhash	opera	trans	balan
1	1	0767c864cef0334f27473902eb9868e7	bdc9065d20a4cd037bb1a7538486403e	2	0	0
2	2	bdc9065d20a4cd037bb1a7538486403e	6619f4809d73a267a4b9ac554bb4523a	1	5	5
3	3	6619f4809d73a267a4b9ac554bb4523a	4d186321c1a7f0f354b297e8914ab240	1	5	5

Two yellow arrows point from the 'prevhash' column of the second row to the 'newhash' column of the third row, illustrating the chain connection between blocks.

Предыдущий хэш связан с новым хэшем.

Они всегда одни и те же.

Блок для выполнения теста работы PoW и добычи новых блоков. (**PoW**)



Обязательная зависимость: **NewBlock**. Представьте этот блок перед использованием.

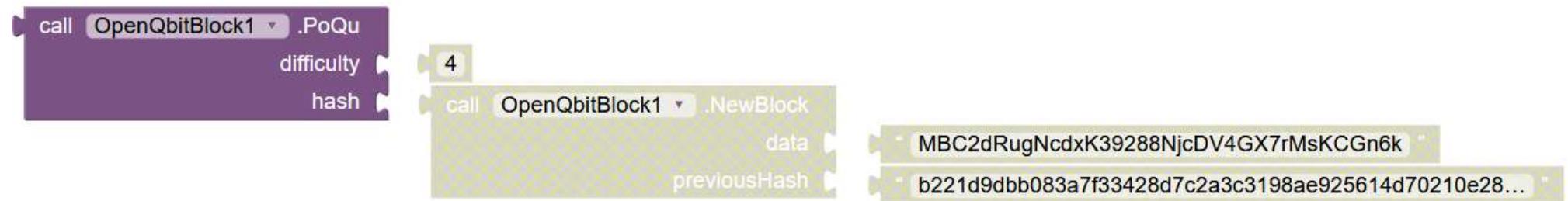
Входные параметры: **сложность** <Integer>, хэш <Принудительная зависимость>.

Выходные параметры: Дает в качестве выхода хэш "SHA256", состоящий из #Number of "zeros", заданного в сложности входного параметра.

Описание: Этот блок выполняет действие, называемое "добычей" нового блока - это то, что мы ранее описывали как процесс PoW (Proof of Work), см. раздел "Что такое доказательство кванта (PQu)?

Добыча или запуск PoW - это понятие, при котором узлам дается задание на решение математической головоломки. Тот, кто решает его первым в случае Mini BlocklyChain, получает возможность продолжить процесс, чтобы быть выбранным победителем, чтобы иметь возможность обрабатывать очередь транзакций, которая ждет обработки.

Блок для выполнения теста работы PoW и добычи новых блоков. (**PoQu**)



Обязательная зависимость: **NewBlock**. Представьте этот блок перед использованием.

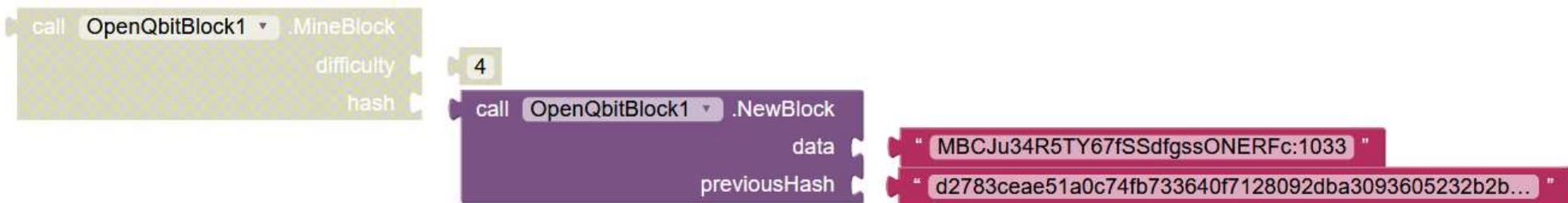
Входные параметры: **сложность** <Integer>, хэш <Принудительная зависимость>.

Выходные параметры: Результат в виде числа "Магическое число" и случайного квантового числа, которое находится в диапазоне 0 и 1 16-значного десятичного типа, например (0. 5843012986202495), и хэша "SHA256", состоящего из #Number of "zeros", заданного в сложности входного параметра.

Описание: Этот блок выполняет процесс "добычи" нового блока, это то, что мы ранее описывали как процесс PoW (Proof of Work), но этот процесс вызывает функцию случайного квантового числа, генерируемого после вычисления числа "nonce", подходящего для удовлетворения уровня сложности, который может быть в диапазоне минимум 1, максимум 5. См. раздел Что такое "Доказательство квантового числа" (PQu - Proof Quantum)?

Добыча или запуск PoW или PoQu - это понятие, при котором узламдается задание на решение математической головоломки. Узел, решающий ее первым в случае любой системы блок-цепочки, получает возможность продолжить процесс, чтобы быть выбранным в качестве победителя, который сможет обработать очередь транзакций, ожидающую обработки.

Блок для создания **новых** блоков (**NewBlock**).

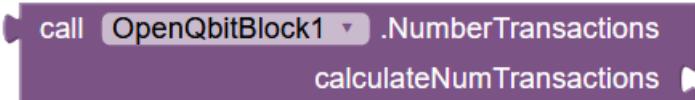


Входные параметры: **данные** <String>, **previousHash** <String>.

Выходные параметры: он выдает хэш, вычисленный как: **SHA256** (**данные** (входные параметры) + **previousHash** (входной параметр) + **IMEI** (внутренний параметр) + **MerkleRoot** (внутренний параметр)).

Описание: Этот блок выполняет создание нового блока для обработки. На выходе выдается хэш "SHA256", состоящий из строки входных параметров в случае изменения данных в зависимости от конструкции системы, а предыдущий хэш основан на хэше предыдущей строки транзакции, которая уже была обработана и хранится в блочной блочной системе Mini BlocklyChain, это переменные параметры, есть два внутренних системных параметра, которые являются фиксированными и обеспечивают целостность информации и системы, эти два внутренних параметра являются уникальным идентификатором мобильного телефона и хэшем дерева мерклетов очереди транзакций, которые обрабатываются.

Блок общих локальных транзакций узла (**NumberTransactions**)



Обязательная зависимость: Блок (**AddKeyValueArrayStoS**). Представьте этот блок перед использованием.

Входные параметры: < Обязательная зависимость>.

Выходные параметры: возвращает узлу общее количество локальных транзакций.

Описание: Предоставляет общую сумму транзакций, которая будет применена только к локальным счетам узла.

Блок для чтения адреса получателя в двоичном файле. (**RecieipientLoadKeyValuePair**)

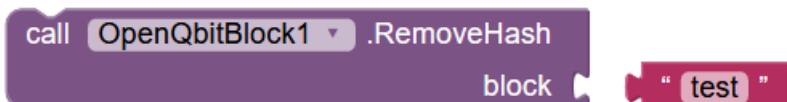


Входные параметры: Не применимо.

Выходные параметры: Возвращает закрытый и открытый ключ в формате Base64.

Описание: Получает частный и публичный адрес получателя из двоичного файла в качестве входного параметра.

Блок для удаления элемента во внутренней временной цепочке (**RemoveHash**).

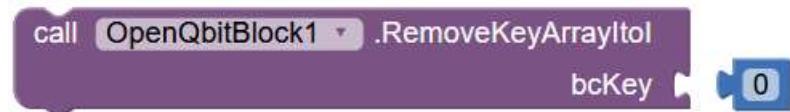


Входные параметры: блок <Строка>.

Выходные параметры: Не применимо.

Описание: Получает частный и публичный адрес получателя из двоичного файла в качестве входного параметра.

Блок для удаления элемента во внутреннем временном устройстве "Itol_UTXO" (RemoveKeyArrayItol)

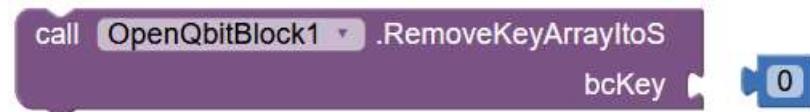


Входные параметры: **bcKey < Целый >**.

Выходные параметры: Не применимо, тип элемента delete <Integer>.

Описание: Удаляется элемент, связанный с числовой меткой внутренней временной договоренности "Itol_UTXO".

Блок для удаления элемента во внутренней временной схеме "ItoS_UTXO" (RemoveKeyArrayItoS).

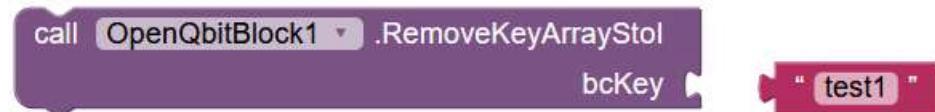


Входные параметры: **bcKey < Целый >**.

Выходные параметры: Не применимо, тип удаляемого элемента <String>.

Описание: Удаляется элемент, связанный с числовой меткой внутренней временной договоренности "ItoS_UTXO".

Блок удаления элемента во внутреннем временном устройстве "Stol_UTXO" (RemoveKeyArrayStol)

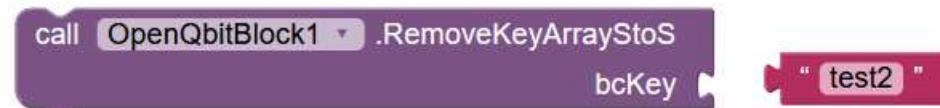


Входные параметры: **bcKey <Строка>**.

Выходные параметры: Не применимо, тип элемента delete <Integer>.

Описание: Удаляется элемент, связанный с числовой меткой внутренней временной расстановки "Stol_UTXO".

Блок удаления элемента во внутреннем временном устройстве "StoS_UTXO" (RemoveKeyArrayStoS)

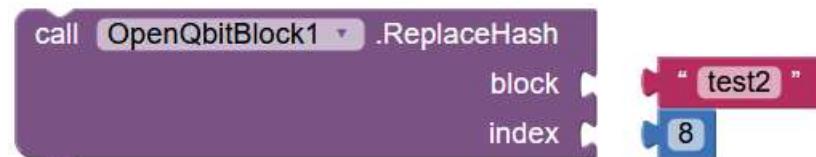


Входные параметры: **bcKey** <Строка>.

Выходные параметры: Не применимо, тип удаляемого элемента <String>.

Описание: Удалить элемент наклейки внутренней временной расстановки "StoS_UTXO".

Блок для замены значения внутренней временной договоренности "цепь" (ReplaceHash).



Входные параметры: **блок** <Строка>, **индекс** <Целый>.

Выходные параметры: Не применимо.

Описание: Элемент, связанный с индексом "индекс", заменяется значением метки "блок" во временном внутреннем расположении "цепочки".

Блокировка для запуска (**отправки**) новой транзакции (SendTrasaction).



Входные параметры: **изAddrMBC** <Строка>, **вAddrMBC** <Строка>, **значение** <Целый>.

Выходные параметры: возвращает значение "True", если транзакция была отправлена успешно, или "False", если она не была отправлена успешно.

Описание: Блок, который преобразует адрес отправителя и получателя в двоичный формат и прикрепляется к очереди транзакций, подлежащих обработке.

Блокировка чтения адреса отправителя в двоичном файле. (**SenderLoadKeyPair**)

 call OpenQbitBlock1 .SenderLoadKeyPair

Входные параметры: **Не применимо**.

Выходные параметры: Возвращает закрытый и открытый ключ в формате Base64.

Описание: Получает частный и публичный адрес отправителя из двоичного файла в качестве входного параметра.

Блок для получения номера элемента временного расположения "цепочки" (**SizeBlockList**).

 call OpenQbitBlock1 .SizeBlockList

Входные параметры: **Не применимо**.

Выходные параметры: Возвращает номер элемента внутреннего массива "chain".

Описание: Получает номер элемента внутреннего массива "chain".

Блок для получения номера элемента временной расстановки "Itol_UTXO" (**SizeItol**)

 call OpenQbitBlock1 .SizeItol

Входные параметры: **Не применимо**.

Выходные параметры: Возвращает номер элемента внутреннего массива "Itol_UTXO".

Описание: Получает номер элемента внутреннего массива "Itol_UTXO".

Блок для получения номера элемента временной расстановки "ItoS_UTXO" (**SizeItoS**)

 call OpenQbitBlock1 .SizeItoS

Входные параметры: **Не применимо**.

Выходные параметры: Возвращает номер элемента внутреннего массива "ItoS_UTXO".

Описание: Получает номер элемента внутреннего массива "ItoS_UTXO".

Блок для получения номера элемента временной расстановки "Stol_UTXO" (SizeStol)

call OpenQbitBlock1 .SizeStol

Входные параметры: Не применимо.

Выходные параметры: Возвращает номер элемента внутреннего массива "Stol_UTXO".

Описание: Получает номер элемента внутреннего массива "Stol_UTXO".

Блок для получения номера элемента временной расстановки "StoS_UTXO" (SizeStoS)

call OpenQbitBlock1 .SizeStoS

Входные параметры: Не применимо.

Выходные параметры: Возвращает номер элемента внутреннего массива "StoS_UTXO".

Описание: Получает номер элемента внутреннего массива "StoS_UTXO".

Блокировка создания транзакции с дополнительными значениями (Сделка).



Входные параметры: activeValue < Целый>, fromAddrMBC < Стока>, toAddrMBC < Стока>, inputValue < Стока массива>.

Выходные параметры: Даёт нам, адреса в двоичном формате и значение inputValue, преобразованное в строку 'String', и даёт нам хэш "SHA256" значения ActiveValue.

Описание: Готовит новую транзакцию для обработки узлами.

Блок проверки адреса мини-блочной цепи (**ValidateAddMiniBlocklyChain**)

call OpenQbitBlock1 .ValidateAddrMiniBlocklyChain
addrValidate " MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k "

Входные параметры: Адреса пользователей в формате Mini BlocklyChain.

Выходные параметры: возвращает "True", если адрес в правильном формате, или "False", если адрес недействителен.

Описание: Проверяет, правильно ли введен адрес Mini BlocklyChain, в этом блоке применяется алгоритм проверки, был ли адрес создан с помощью механизма создания адреса, который будет использоваться в системе Mini BlocklyChain.

Блок подтверждения адреса биткойна. (**ValidateAddBitcoin**)

call OpenQbitBlock1 .ValidateAddressBitcoin
addr " 12dRugNcdxK39288NjcDV4GX7rMsKCGn6k "

Входные параметры: **addr** < String> , адреса пользователей в формате Bitcoin (принимает адреса Bitcoin с начальным идентификатором "1", адреса с идентификатором "3" не применимы).

Выходные параметры: возвращает "True", если адрес в правильном формате, или "False", если адрес недействителен.

Описание: Проверяет, правильно ли введён адрес Биткойна, в данном блоке применяется алгоритм проверки, был ли адрес создан с помощью механизма создания адреса, который будет использоваться в системе Биткойна.

Блок проверки электронной подписи для текущей транзакции. (**VerifySignature**).

call OpenQbitBlock1 .VerifySignature

Входные параметры: Не применимо.

Выходные параметры: возвращает "True", если проверка достоверна, или "False", если проверка недействительна.

Описание: Получает информацию о проверке цифровой подписи, которую отправитель должен был видеть в процессе отправки транзакции, которую вы хотите совершить. В этом процессе проверки проверяется, что отправленное значение источника не было

изменено в канале, по которому была отправлена транзакция, а также проверяется получатель, к которому должна быть применена транзакция

20. Использование блоков для базы данных SQLite (версия MiniSQLite)

В этом разделе мы увидим, как с помощью блоков выполнять две основные операции, которые нас интересуют для функциональности системы Mini BlocklyChain, а именно "INSERT" данных в цепочку блоков и запрашивать их.

ПРИМЕЧАНИЕ: Транзакции в базе данных SQLite отличаются от транзакций, посылаемых узлами, которые будут применяться в системе Mini BlocklyChain.

Транзакции в базе данных основаны на модели CRUD (Create, Read, Update and Delete) и представляют собой процессы, которые могут выполняться с внешними или внутренними данными только в базе данных SQLite. В блочных системах используются в основном процессы Создания и Чтения, для безопасности отбрасываются процессы обновления или удаления и многое другое, где хранится цепочка блоков, что дает целостную безопасность системы.

С другой стороны, транзакции, посылаемые узлами, относятся ко всем процессам, которые подразумевают принятие решения об отправке того или иного актива между участниками (узлами) системы Mini BlocklyChain, этот тип транзакций включает в себя различные процессы для его применения, это может быть создание цифрового адреса, цифровая подпись, проверка подписи, процесс в базе данных SQLite, среди прочих процессов.

Начнем с определения и использования блоков базы данных SQLite версии MiniSQLite эта версия интегрирована всего на 8 блоков. У вас есть полная версия для работы с данными в базе данных SQLite, однако для практических целей достаточно системы Mini BlocklyChain с версией MiniSQLite.

В случае, если вы хотите ознакомиться со всеми компонентами, их использованием и описанием, смотрите приложение "Расширенные блоки для базы данных SQLite".

Блоки версий MiniSQLite:

Блокировка запуска некой транзакции в базе данных SQLite (**BeginTransaction**)

call OpenQbitQSQLite1 .BeginTransaction

Обязательные единицы: Блок (**ImportDatabase**), Блок (**OpenDatabase**).

Параметры ввода: **Использовать до < Обязательная зависимость(и)>**.

Выходные параметры: Не применимо, запускает транзакцию в базе данных SQLite.

Описание: Блок, который запускает процесс в БД SQLite, должен сначала выполнить блок импорта БД (**ImportDatabase**) и блок открытия БД (**OpenDatabase**).

Блок для фиксации в SQLite. (**CommitTrasaction**)

call OpenQbitQSQLite1 .CommitTransaction

Необходимые зависимости: Блок (**BeginTransaction**), Блок (**ImportDatabase**), Блок (**OpenDatabase**).

Параметры ввода: Использовать до < Обязательная зависимость(и)>.

Выходные параметры: Не применимо, выполняет **фиксацию** транзакции в базе данных SQLite.

Описание: Блок, который запускает процесс **фиксации** в базе данных SQLite, вы должны сначала выполнить блок импорта базы данных (**ImportDatabase**) и блок открытия базы данных (**OpenDatabase**).

Блок для закрытия импортированной или экспортированной базы данных SQLite (**CloseDatabase**)

call OpenQbitQSQLite1 .CloseDatabase

Обязательные единицы: Блок (**ImportDatabase**), Блок (**OpenDatabase**).

Параметры ввода: Использовать до < Обязательная зависимость(и)>.

Выходные параметры: Не применимо, закрывает базу данных SQLite.

Описание: Блок, который закрывает базу данных SQLite, вы должны сначала выполнить блок импорта базы данных (**ImportDatabase**) и блок открытия базы данных (**OpenDatabase**).

Блок для экспорта базы данных SQLite (**ExportDatabase**).

call OpenQbitQSQLite1 .ExportDatabase

fileName

" mbcExport.sqlite "

Обязательные единицы: Блок (**ImportDatabase**), Блок (**OpenDatabase**).

Входные параметры: **fileName < String >** введите путь, по которому можно найти базу данных, уже созданную в формате SQLite.

Использовать до < Обязательная зависимость(и) >

Выходные параметры: Экспорт в базу данных SQLite.

Описание: Блок, который экспортирует базу данных SQLite, вы должны сначала выполнить блок импорта базы данных (**ImportDatabase**) и блок открытия базы данных (**OpenDatabase**).

Блок для импорта базы данных SQLite. (**ImportDatabase**)

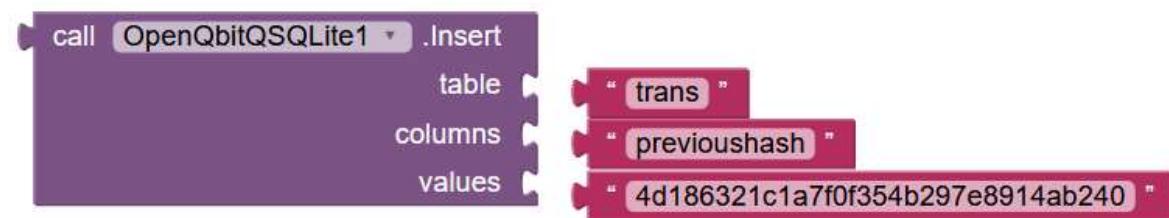


Входные параметры: **fileName < String >** введите путь, по которому можно найти базу данных, уже созданную в формате SQLite.

Выходные параметры: запускает базу данных SQLite для выполнения транзакций.

Описание: Блок, который запускает процесс в БД SQLite, должен сначала выполнить блок импорта БД (**ImportDatabase**) и блок открытия БД (**OpenDatabase**).

Блок для вставки данных в базу данных SQLite. (**Вставить**)



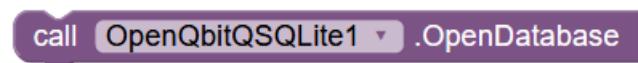
Обязательные единицы: Блок (**ImportDatabase**), Блок (**OpenDatabase**).

Входные параметры: **таблица <Строка>, столбцы <Строка>, значения <Строка>**, **Использовать до <Принудительная зависимость(и)>**.

Выходные параметры: вставляет транзакцию в базу данных SQLite.

Описание: Блок, который вставляет данные в базу данных SQLite, вы должны сначала выполнить блок импорта базы данных (**ImportDatabase**) и блок открытия базы данных (**OpenDatabase**).

Блок для открытия базы данных SQLite (**OpenDatabase**)



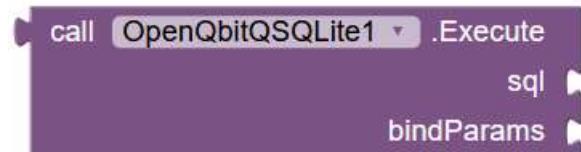
Обязательная единица(и): Блок (**ImportDatabase**).

Параметры ввода: **Использовать до** < Обязательная зависимость(и)>.

Выходные параметры: неприменимо, запускать или открывать базу данных SQLite для выполнения транзакций.

Описание: Блок, который запускает базу данных SQLite, должен быть раньше.

Блок для запроса данных в SQLite (**Execute**).



Обязательные единицы: Блок (**ImportDatabase**), Блок (**OpenDatabase**).

Параметры ввода: **sql** < Стока>, **bindParams** < Стока>, **Использовать перед** < Обязательной зависимостью(ами)>.

Выходные параметры: SQL-оператор выполняется для создания транзакции в базе данных SQLite.

Описание: Блок, выполняющий SQL-запросы в БД SQLite, должен сначала выполнить блок импорта БД (**ImportDatabase**) и блок открытия БД (**OpenDatabase**).

21. Определение и использование блоков безопасности.

В этой сессии мы рассмотрим использование блоков, которые обеспечивают нам уровни безопасности для сохранения, проверки и передачи транзакций с узлов сети Mini BlocklyChain.

Блоки безопасности основаны на следующем расширении:

- I. Расширение OpenQbitAESEncryption.
- II. Расширение OpenQbitAESDecryption.
- III. Расширение OpenQbitAEToString.
- IV. Расширение OpenQbitEncDecData.
- V. Расширение OpenQbitFileHash.
- VI. Расширение OpenQbitRSA.
- VII. Расширение OpenQbitSSHClient (потребуется).
- VIII. Расширение OpenQbitStringHash.

За исключением расширения OpenQbitSSHClient, которое является обязательным, вышеуказанные расширения являются необязательными для использования при создании публичной или частной сети Mini BlocklyChain.

Тем не менее, для того, чтобы иметь безопасную систему для ваших операций в зависимости от каждого бизнес-приложения, использование расширений, которые являются необязательными, должно применяться по вашему усмотрению.

Например, в случае использования хэша можно использовать ряд опций алгоритма, таких как MD5, SHA1, SHA128, SHA256, SHA512 для строки символов, а также применяется к любому типу файлов в зависимости от информационного потока каждой системы, созданной с помощью Mini BlocklyChain.

Расширение OpenQbitAESEncryption.

Блокировка для шифрования файлов с помощью AES-безопасности. (AESEncryption).



Входные параметры: **pathFileIn** <Строка> , **pathFileOut** <Строка> , **pathFile** <Строка>, **pathFileVI** <Строка> и **passwd** <Строка>. Имена файлов являются произвольными и зависят от дизайна каждой системы.

Выходные параметры: зашифрованный AES-файл, введенный во входной параметр **pathFileIn**.

Описание: Блок, который дает нам три выходных файла, один из которых является исходным файлом, уже зашифрованным алгоритмом AES с 256-битным ключом, два других файла используются для управления расширением для расшифровки и восстановления исходного файла.

Расширение OpenQbitAESDecryption.

Блок для расшифровки файла AES. (**AESDecryption**).



Входные параметры: **pathFileIn** <Строка> , **pathFileOut** <Строка> , **pathFile** <Строка>, **pathFileVI** <Строка> и **passwd** <Строка>. Имена файлов совпадают с именами, полученными в результате работы блока (**AESEncryption**).

Выходные параметры: Оригинальный файл расшифровывается с помощью AES, введенного во входной параметр **pathFileIn**, в этом случае для расшифровки файла он вводится в **pathFileIn** зашифрованного файла, а в **pathFileOut** дает нам оригинальный файл (расшифровывается).

Описание: Блок, который дает нам файл в параметре **pathFileOut**, который будет выводом, расшифрованным алгоритмом AES с 256-битным ключом.

Расширение OpenQbitAESToString.

Это расширение является одноразовым для каждой сессии устройства, т.е. работает только тогда, когда устройство не перезагружается (мобильный телефон), так как значение VI шифрования генерируется временно.

ПРИМЕЧАНИЕ: Если устройство перезагружается, зашифрованная строка не может быть восстановлена.

Блок для временного шифрования символьных строк (**DecrypSecretText**)

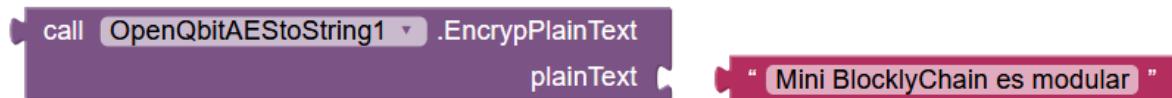


Входные параметры: **secretText** < String >

Выходные параметры: расшифровка исходной символьной строки с помощью AES с 256-битным ключом

Описание: Блок, который дает нам строку символов, является входным параметром, который был введен в блоке (**EncrypPlainText**).

Блок для декодирования строки (**EncrypPlainText**)



Входные параметры: **plainText** < String >

Выходные параметры: зашифрованная AES символьная строка с использованием 256-битного ключа.

Описание: Блок, который передает буквенно-цифровую строку символов, зашифрованную с помощью AES с использованием 256-разрядного цифрового ключа.

Расширение OpenQbitEncDecData.

Специализированный блок шифрования для универсальных баз данных (**EncryptionData**)



Входные параметры: **plainText < String>**

Выходные параметры: зашифрованная AES символьная строка с использованием 256-битного ключа.

Описание: Блок, который передает буквенно-цифровую строку символов, зашифрованную с помощью AES с использованием 256-разрядного цифрового ключа.

Специализированный блок шифрования для универсальных баз данных (**DescriptionData**)



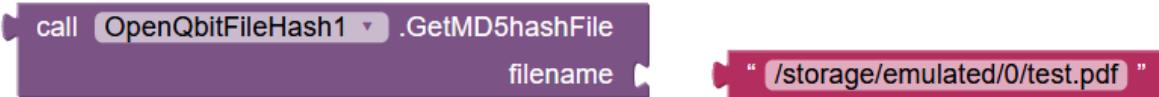
Входные параметры: **plainText < String>**

Выходные параметры: зашифрованная AES символьная строка с использованием 256-битного ключа.

Описание: Блок, который передает буквенно-цифровую строку символов, зашифрованную с помощью AES с использованием 256-разрядного цифрового ключа.

Расширение OpenQbitFileHash.

Блокировать для генерации MD5 из файла (**GetMD5hashFile**)

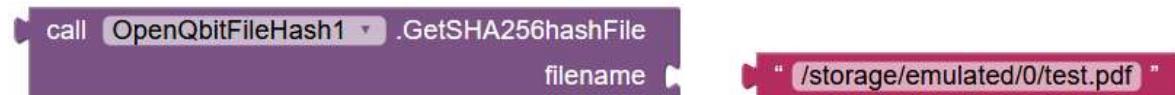


Входные параметры: **имя файла <стринг>**

Выходные параметры: выдает хэш-файл MD5.

Описание: Блок для создания MD5-хэша файла, заданного во входном параметре.

Блок для генерации SHA256 из файла (**GetSHA256hashFile**)

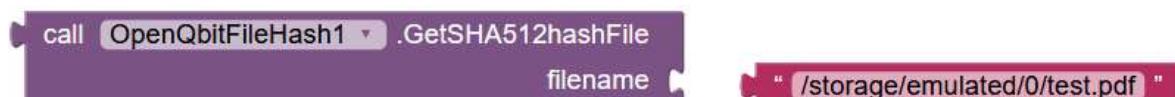


Входные параметры: **имя файла** <стринг>

Параметры вывода: выдаёт хэш архива SHA256.

Описание: Блок для создания хэша SHA256 файла, заданного во входном параметре.

Блок для генерации SHA512 из файла (**GetSHA512hashFile**)

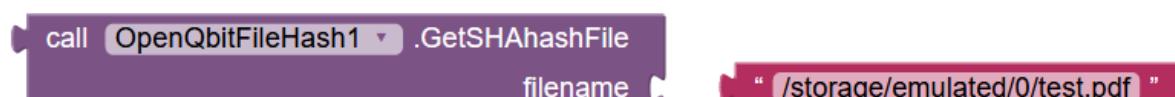


Входные параметры: **имя файла** <стринг>

Параметры вывода: выдаёт хэш архива SHA256.

Описание: Блок для создания хэша SHA256 файла, заданного во входном параметре.

Блок для генерации SHA1 из файла (**GetSHA1hashFile**)



Входные параметры: **имя файла** <стринг>

Выходные параметры: выдаёт хэш архива SHA1.

Описание: Блок для создания SHA1-хэша матрицы во входном параметре.

Расширение OpenQbitRSA.

Блок для **расшифровки** строки с помощью RSA (**Decrypt**)



Необходимые зависимости: Блок (**шифрование**), Блок (**OpenFromDiskPrivateKey**), Блок (**OpenFromDiskPublicKey**).

Входные параметры: **результат** < Стока>

Выходные параметры: Стока символов расшифровывается с помощью RSA.

Описание: Блок, который дает нам строку буквенно-цифровых символов, расшифрованных с помощью ключа того размера, который использовался в блоке (**GenKeyValuePair**).

Блок для шифрования строки с помощью RSA (**Encrypt**)



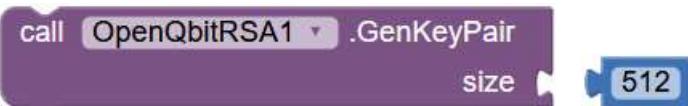
Необходимые блоки: Блок (**GenKeyValuePair**), Блок (**SaveFromDiskPrivateKey**), Блок (**SaveFromDiskPublicKey**).

Входные параметры: **обычный** < Стока>

Выходные параметры: зашифрованная RSA символьная строка

Описание: Блок, который дает нам строку буквенно-цифровых символов, расшифрованных с помощью ключа того размера, который использовался в блоке (**GenKeyValuePair**).

Блок для **расшифровки** строки с помощью RSA (**Decrypt**)



Входные параметры: **размер** < Целый>

Выходные параметры: Не применимо.

Описание: Блок для генерации закрытого и открытого ключа на основе выбранного размера.

Блокировка для получения приватного ключа (**GetPrivateKeyString**)



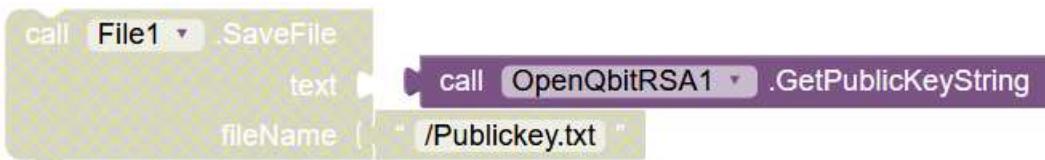
Обязательные блоки: Блок (**GenKeyPair**), Блок (**GenKeyPair**), Блок (**Файл**).

Входные параметры: **Не применимо**.

Выходные параметры: Файл с зашифрованной RSA символьной строкой (закрытый ключ)

Описание: Блок, который дает нам буквенно-цифровую строку, представляющую собой закрытый ключ, зашифрованный с помощью ключа размера, который использовался в блоке (**GenKeyPair**).

Блокировка для получения закрытого ключа (**GetPublicKeyString**)



Обязательные блоки: Блок (**GenKeyPair**), Блок (**GenKeyPair**), Блок (**Файл**).

Входные параметры: **Не применимо**.

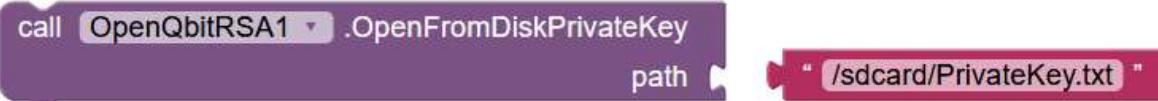
Выходные параметры: Файл с зашифрованной RSA строкой (открытый ключ)

Описание: Блок, который дает нам буквенно-цифровую строку, представляющую собой открытый ключ, зашифрованный с использованием размера ключа, используемого в блоке (**GenKeyPair**).

ПРИМЕЧАНИЕ: В предыдущих блоках (**GetPrivateKeyString**) и (**GetPublicKeyString**) в зависимостях мы будем использовать общий блок (**Файл**) приложения App Inventor паллетной сессии "Хранение".

Такой способ хранения закрытого и открытого ключа с помощью блока (**файла**) может помочь нам лучше манипулировать информацией.

Блокировка чтения приватного ключа из файла (**OpenFromDiskPrivateKey**).



Необходимые зависимости: Блок (**SaveFromDiskPrivateKey**) или Блок (**GetPrivateKeyString**).

Входные параметры: **путь <Строка>**

Выходные параметры: Системная загрузка строки символов с зашифрованным RSA приватным ключом.

Описание: Блок, который дает нам зашифрованную буквенно-цифровую символьную строку закрытого ключа, хранящегося в указанном пути к файлу.

Блокировка чтения открытого ключа из файла (**OpenFromDiskPublicKey**)



Обязательные блоки: Блок (**SaveFromDiskPublicKey**) или Блок (**GetPublicKeyString**).

Входные параметры: **путь <Строка>**

Выходные параметры: Загрузить в систему RSA строку символов с открытым ключом, зашифрованную.

Описание: Блок, который дает нам зашифрованную буквенно-цифровую строку открытого ключа, хранящегося в указанном пути к файлу.

Блок для сохранения приватного ключа к файлу (**SaveToDiskPrivateKey**).



Обязательный блок (ы): Блок (**GenKeyPair**).

Входные параметры: **обычный** <Строка>

Выходные параметры: Файл с зашифрованной RSA строкой. (закрытый ключ)

Описание: Блок, который дает нам файл с буквенно-цифровой строкой, зашифрованный с помощью закрытого ключа размера, используемого в блоке (**GenKeyValuePair**).

Блок для сохранения приватного ключа к файлу (**SaveToDiskPublicKey**).



Обязательный блок (ы): Блок (**GenKeyValuePair**).

Входные параметры: **обычный** <Строка>

Выходные параметры: Файл с зашифрованной RSA строкой. (публичный ключ)

Описание: Блок, который дает нам файл с буквенно-цифровой строкой, зашифрованный с помощью открытого ключа размером, который использовался в блоке (**GenKeyValuePair**).

Расширение OpenQbitSSHClient.

Коннектор Блок SSH клиента (**ConnectorMiniBlocklyChain**)

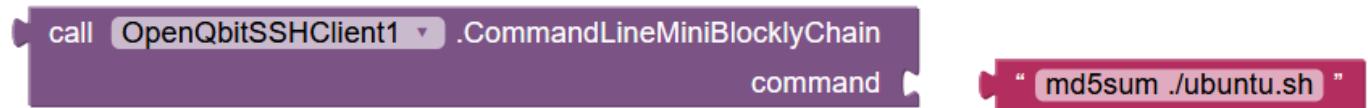


Входные параметры: имя пользователя <string>, пароль <string>, хост <string>, порт<integer>.

Выходные параметры: если соединение с ssh-сервером терминала Termux успешно, то выдается сообщение; "**Connect SSH**", если не успешно, выдается сообщение **NULL**.

Описание: Коммуникационный блок для подключения Mini BlocklyChain к терминалу Termux, через протокол связи SSH (Secure Shell).

Блок для выполнения команд в Linux терминале Termux (**CommandLineMiniBlocklyChain**).



Входные параметры: **команда** <строка>

Выходные параметры: Переменные данные, в зависимости от выполняемой команды или программы.

Описание: Блок выполнения команд в терминале Termux - необходимое условие для установления соединения с блоком (**ConnectorMiniBlocklyChain**) может выполнять все виды команд в режиме онлайн и/или получать конкретные данные о выполнении от скриптов или программ, имеющих интерфейс CLI (Command-Line Interface) в режиме онлайн.

ОтключитьMiniBlocklyChainSSH.

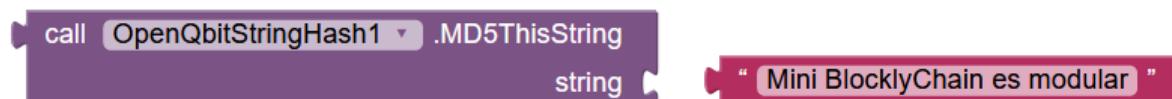


Входные и выходные параметры: Не применимо (нет)

Описание: Блок для закрытия сеанса SSH.

Расширение OpenQbitStringHash.

Блок для генерации MD5-строки (**MD5ThisString**)



Входные параметры: строка

Параметры вывода: выдает хэш MD5.

Описание: Блок для создания MD5-хэша строки, заданного во входном параметре.

Блок для генерации строки символов SHA256 (**SHA256ThisString**)

call OpenQbitStringHash1 .SHA256ThisString
string

“ Mini BlocklyChain es modular ”

Входные параметры: строка

Выходные параметры: выдает хэш SHA256.

Описание: Блок для создания хэша SHA256 строки, заданного во входном параметре.

Блок для генерации строки SHA512 (SHA512ThisString)

call OpenQbitStringHash1 .SHA512ThisString
string

“ Mini BlocklyChain es modular ”

Входные параметры: строка

Параметры выхода: выдает хэш SHA512.

Описание: Блок для создания хэша SHA512 строки, заданного во входном параметре.

Блок для генерации строки символов SHA1 (SHAThisString)

call OpenQbitStringHash1 .SHAThisString
string

“ Mini BlocklyChain es modular ”

Входные параметры: строка

Выходные параметры: выдает хэш SHA1.

Описание: Блок для создания SHA1-хэша строки, заданного во входном параметре.

22. Установка параметров безопасности в Mini BlocklyChain.

Параметры безопасности разделены на три компонента любой системы, которая спроектирована и применяется в следующих компонентах:

- a. База данных Redis (резервная сеть связи)
 - b. Система синхронизации Peer to Peer Syncthing System
 - c. Интеграция безопасности для защиты базы данных SQLite
 - d. Среда разработчика для интеграции модулей
-
- a. База данных Redis (резервная сеть связи)

Переименование опасных команд, дополнительная встроенная функция безопасности Redis включает в себя переименование или отключение некоторых команд, которые считаются опасными.

При выполнении неавторизованными пользователями эти команды могут быть использованы для перенастройки, уничтожения или удаления их данных. Как и пароль аутентификации, команды переименования или отключения настраиваются в том же разделе **БЕЗОПАСНОСТЬ** файла `/etc/redis/redis.conf`.

Некоторые из команд, считающихся опасными: **FLUSHDB**, **FLUSHALL**, **KEYS**, **PEXPIRE**, **DEL**, **CONFIG**, **SHUTDOWN**, **BGREWRITEAOF**, **BGSAVE**, **SAVE**, **SPOP**, **SREM**, **RENAME** и **DEBUG**. Это не полный список, но переименование или отключение всех команд из этого списка является хорошим началом для улучшения безопасности вашего сервера Redis.

В зависимости от ваших конкретных потребностей или потребностей вашего сайта, вы должны переименовать или отключить команду. Если вы знаете, что никогда не будете использовать команду, которой можно манипулировать, вы можете отключить ее. С другой стороны, вы можете захотеть переименовать его.

Чтобы включить или отключить команды Redis, повторно откройте файл конфигурации:

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

Внимание: Следующие шаги по отключению и переименованию команд являются примерами. Вы должны выбрать только отключить или переименовать команды, которые применяются к вам. Вы можете просмотреть полный список команд и определить, как они могут быть неправильно использованы в redis.io/commands.

Чтобы отключить команду, просто переименуйте ее так, чтобы она стала пустой строкой (символизируемой парой кавычек без символов между ними), как показано ниже:

/etc/redis/redis.conf

```
.
.
.
# Также можно полностью убить команду, переименовав ее в
# пустая строка:
#
переименование-команда FLUSHDB ""
переименование-команда FLUSHALL ""
переименование команды DEBUG ""
```

Чтобы переименовать команду, дайте ей другое имя, как показано в примерах ниже. Переименованные команды должно быть трудно угадать другим, но легко запомнить.

/etc/redis/redis.conf

```
.
.
.
# rename-command CONFIG ""
переименование команды SHUTDOWN SHUTDOWN_MENOT
переименование-команда CONFIG ASC12_CONFIG
```

Сохраните изменения и закройте файл.

После переименования команды, применить изменение, перезапустив Redis:

- sudo systemctl restart redis.service

Чтобы проверить новую команду, введите командную строку Redis:

- redis-cli

Затем выполните аутентификацию:

- auth your_redis_password

Выход

OK

Как и в предыдущем примере, предположим, что вы переименовали команду CONFIG в ASC12_CONFIG. Сначала попробуйте использовать оригинальную команду CONFIG. Потому что ты переименовал его, это не должно работать:

- получить требуемую конфигурацию

Выход

(ошибка) ERR неизвестная команда 'config'.

Однако, переименованная команда может быть вызвана успешно. Он не чувствителен к регистру:

- `asc12_config` получить требуемый эстакад

Выход

- 1) "эстакада"
- 2) "ваше_редис_пароль".

Наконец, вы сможете закрыть перекройку:

- выход

Обратите внимание, что если вы уже используете командную строку Redis и перезапустите Redis, вам нужно будет пройти повторную аутентификацию. В противном случае при вводе команды эта ошибка будет отображена:

Выход

Нет необходимости в аутентификации.

b. Система синхронизации Peer to Peer Syncing System

При использовании системы "Peer to Peer" между узлами в коммуникационной сети применяются следующие три элемента

-Приватный: информация хранится только на ваших компьютерах. Не существует центрального сервера, который может быть взломан (легально или нелегально).

-Encrypted: Все коммуникации защищены с помощью протокола TLS (Transport Layer Security - безопасность транспортного уровня); криптографический протокол, который включает в себя идеальную последовательность, чтобы предотвратить доступ к вашей информации кому-либо, кому вы не доверяете.

-Аутентифицировано: каждый узел идентифицируется с помощью сильного криптографического сертификата. Только узлы, которые вы явно разрешили, могут подключаться к вашей информации.

<https://docs.syncing.net/users/security.html>

С помощью онлайн-команды SyncingManager:

<https://github.com/classicsc/syncingmanager>

c. Интеграция безопасности для защиты базы данных SQLite

При использовании расширения (**OpenQbitSQLite**) или в его случае расширения (**ConnectorSSHClient**) для использования SQLite CLI оба расширения могут быть объединены с расширением безопасности AES (**OpenQbitEncDecData**).

См. Приложение "Пример создания системы Mini BlocklyChain".

d. Среда разработчика для интеграции модулей

Реализация безопасности в среде разработки может быть осуществлена с помощью двух процессов:

- Ограничение использования библиотек OpenJDK.
- Использование ограничено авторизованными узлами системы.

23. Приложение "Создание баз данных KeyStore & PublicKeys".

KeyStore - это хранилище сертификатов безопасности, либо сертификатов авторизации, либо сертификатов открытого ключа, паролей или общих ключей безопасности, таких как соответствующие частные ключи (адреса) пользователя, которые используются для создания или обработки транзакций.

Это зашифрованная база данных, так что только ее владелец может ею воспользоваться. Обычно это локальный тип, однако в системе Mini BloclyChain это безопасная база данных, но она является общей и распространяется на всех узлах, это в основном связано с простой причиной, все узлы должны знать адреса всех пользователей (публичные адреса), частные адреса всегда являются локальными и имеют уникальное и исключительное использование каждого пользователя, однако, при выполнении входа (депозита) или выхода (расходов) транзакции каждая транзакция всегда имеет по крайней мере три компонента при создании: исходный адрес, адрес назначения и актива или стоимости, которая отправляется.

Для создания нашего KeyStore мы должны будем выполнить следующие шаги и требования.

Первым требованием является наличие типа хранилища, где они будут храниться, в нашем случае мы будем использовать базу данных SQLite и создадим два KeyStore, один с личными ключами пользователя, который будет локальным, и он будет защищен "зашифрованной" информацией, а другая база данных, которая будет хранить открытые ключи, эта база будет совместно использоваться всеми узлами сети, база, которая будет совместно использоваться в сети, также будет зашифрована, однако эта база будет иметь пароль, который смогут совместно использовать только члены (узлы) сети.

Вторым фундаментальным требованием является процесс шифрования информации, это будет сделано с помощью специализированного расширения для использования в общей базе данных под названием (**OpenQbitEncDecData**), которая использует алгоритм AES.

Расширение (**OpenQbitEncDecData**) функционально для проверки унитарных элементов блочной цепочки, однако, в случае необходимости проверки полной целостности блочной цепочки это будет неэффективно, поэтому мы также будем осуществлять шифрование копии, но в данном случае это будет осуществляться через расширения: (**OpenQbitAESEncryption**) и (**OpenQbitAESDecryption**), которые работают с файлом, а не со ссылками на данные.

ПРИМЕЧАНИЕ: Для правильной работы базы данных **KeyStore** на терминале Termux должен быть запущен SSH-сервер. Выполните команду в терминале:

sshd

Расширения, основанные на алгоритме AES, могут быть использованы для любого типа данных или файла, и этот алгоритм был выбран, так как он является единственным, который доказал, что является доказательством против атак, основанных на квантовых вычислениях.

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	Minado 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

Приведенная выше таблица относится к Национальным академиям наук, инженерных наук и медицины.

<https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects>

Описание

Квантовая механика, подполе физики, описывающее поведение очень маленьких частиц (квантов), является основой новой вычислительной парадигмы. Впервые предложенная в 1980-х годах в качестве способа совершенствования вычислительного моделирования квантовых систем, область квантовых вычислений в последнее время привлекает значительное внимание в связи с прогрессом в строительстве малоразмерных устройств. Однако для того, чтобы практический квантовый компьютер

мог быть реализован в больших масштабах, потребуются значительные технические усовершенствования.

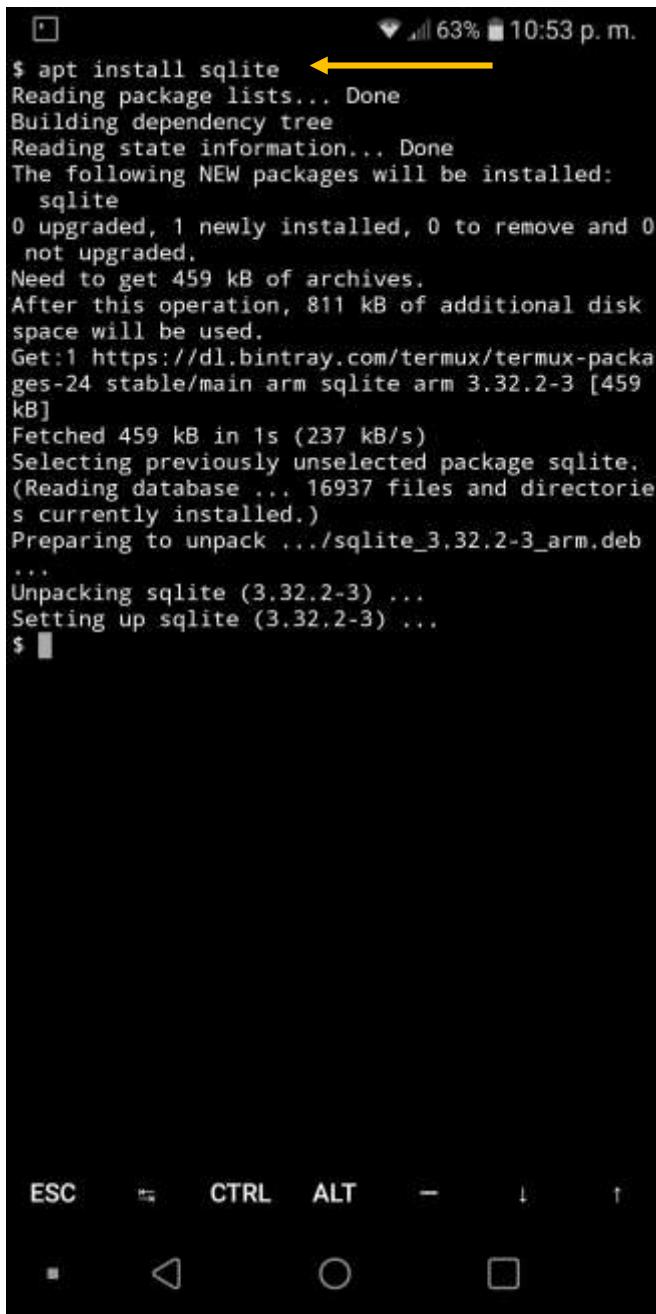
Quantum Computing: Progress and Prospects (Квантовые вычисления: прогресс и перспективы) дает представление об этой области, в том числе об уникальных характеристиках и ограничениях технологии, а также оценивает целесообразность и последствия создания функционального квантового компьютера, способного решать реальные проблемы. В этом докладе рассматриваются требования к аппаратному и программному обеспечению, квантовые алгоритмы, движущие силы прогресса в области квантовых вычислений и квантовых устройств, контрольные показатели, связанные с соответствующими случаями использования, время и ресурсы, необходимые для этого, а также способы оценки вероятности успеха.

Установка обработчика БД SQLite в терминале TERMUX и тестирование CLI (Command-Line) команды **sqlite3** путем создания БД с именем **keystore.db**.

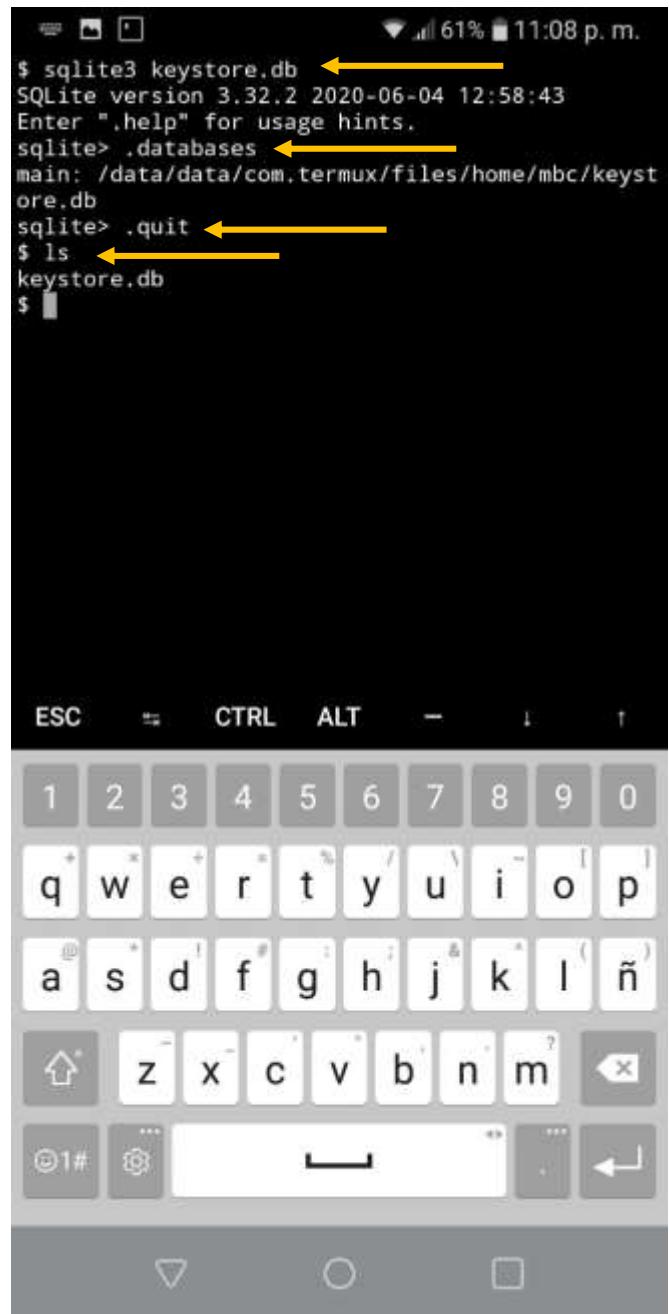
Мы используем команды:

\$ подходящий кв.м.

\$ кв.3 клавишный.дб



```
$ 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 directorie
s 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/mbc/keyst
ore.db
sqlite> .quit
$ ls
keystore.db
$
```

Затем внутри **обработчика sqlite>** выполняем предложение `.databases` для проверки, в каком пути находится база данных, которую мы создаем, затем для выхода и сохранения базы данных даем предложение `.quit`.

ЗАМЕЧАНИЕ: В обоих операторах или командах в **обработчике sqlite>** необходимо сначала поставить точку `"..."` в синтаксисе.

Наконец, мы проверяем, что (пустая) БД уже создана, передавая команду:

\$ ls

Далее мы создадим таблицу, в которой первичные ключи будут храниться в трех различных форматах: шестнадцатеричный, двоичный и адрес ссылки пользователя Mini BlocklyChain, который является публичным ключом соответствующего первичного ключа.

Шифрование данных AES будет применяться только в шестнадцатеричном и двоичном форматах. В случае адреса пользователя addrMBC и псевдонима не потому, что это публичный ключ, который может и должен быть совместно использован по сети для получения транзакций, а также псевдоним для выполнения поиска по этому полю.

Создала таблицу под названием "privatekey" в БД в SQLite (keystore.db).

```
CREATE TABLE privatekey (
    целочисленный первичный ключ ID
    также известный как      НИЧТОЖНЫЙ VARVAR(50)
    addrHexVARCHAR    (65) NOT NULL
    addrMBCVARCHAR   (65) NOT NULL
    addrBinBLOB      НЕ НОЛЬКО
);
```

Выполним предложения в sqlite CLI для создания базы данных keystore.db.

Снова используем командную строку sqlite3 со следующей командой:

\$ kv.3

Это отправит нас внутрь **sqlite>** менеджера **баз данных**, в этом первом мы откроем уже созданную БД, чтобы иметь возможность работать над ней, с помощью следующего предложения внутри менеджера:

```
Sqlite> .open keystore.db
```

Затем мы введем SQL оператор "**CREATE TABLE**" для создания таблицы **приватных ключей**.

Далее показаны два варианта создания одной и той же таблицы, один - через одну строку, в которую включаются все SQL-операторы интегрируемых в нее элементов.

Второй пример - это введение каждого элемента, интегрирующего структуру сегментированным образом.

§ кв.3

SQLite версия 3.32.2 2020-06-20 15:25:24

Введите ".help" для подсказок по использованию.

Подключен к базе данных переходных процессов в памяти.

Используйте ".open FILENAME" для повторного открытия в постоянной базе данных.

sqlite> .open keystore.db

sqlite> CREATE TABLE privatekey (id integer primary key AUTOINCREMENT NOT NULL, он же VARCHAR(50) NOT NULL, addrHex VARCHAR(65) NOT NULL, addrMBC VARCHAR(65) NOT NULL, addrBin BLOB NOT NULL);

sqlite> .quit

Далее показывается создание той же таблицы **приватных ключей**, но именно путем введения SQL-оператора сегментированным способом:

sqlite> CREATE TABLE privatekey (
...> id целочисленный первичный ключ AUTOINCREMENT
...> ... так же известный как VARCHAR(50) NOT NULL,
...> addrHex VARCHAR (65) NOT NULL,
...> addrMBC VARCHAR (65) NOT NULL,
...> addrBin BLOB NOT NULL
...>);

sqlite> .таблицы

закрытый ключ

sqlite> .quit

Два вышеприведенных примера дают один и тот же результат. Позже мы выполняем предложение **.tables**, чтобы проверить, что таблица приватного ключа была создана, в конце мы даем предложение **.quit**, при этом мы уже создали таблицу **приватного ключа** в базе данных SQLite **keystore.db**.

До этого момента у нас уже есть структура базы данных **keystore.db** и ее таблицы закрытых ключей, где закрытые ключи будут храниться в зашифрованном виде.

Это должно показать что-то подобное в узле (мобильном телефоне) с терминалом TERMUX.

```

$ 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
$ 

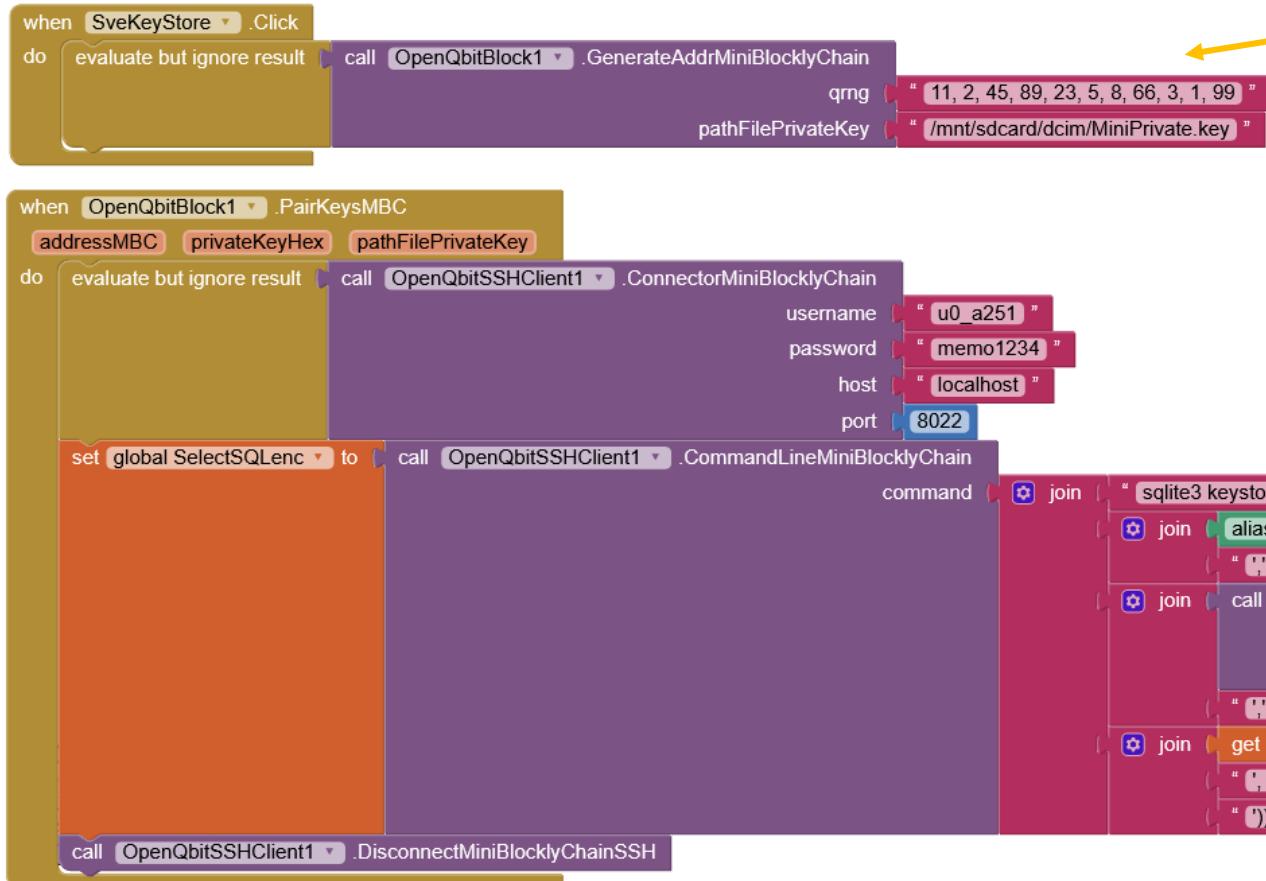
```

SQL-операторы
для создания
новой таблицы

Теперь мы будем использовать некоторые расширения безопасности для вставки данных "закрытого ключа", а затем обратиться к этим данным.

Мы будем использовать блок для генерации нового адреса пользователя (**GenerateAddrMiniBlocklyChain**), этот блок даст нам частный адрес в двух форматах (шестнадцатеричный и двоичный), а также публичный адрес в формате общего адреса МВС. Мы также будем использовать расширение (**OpenQbitSSHClient**) и расширение (**OpenQbitEncDecData**), чтобы посмотреть подробности в разделе "Определение и использование блоков безопасности". В терминале Termux должен работать сервис SSH.

Зашифрованные данные INSERT в базу данных keystore.db

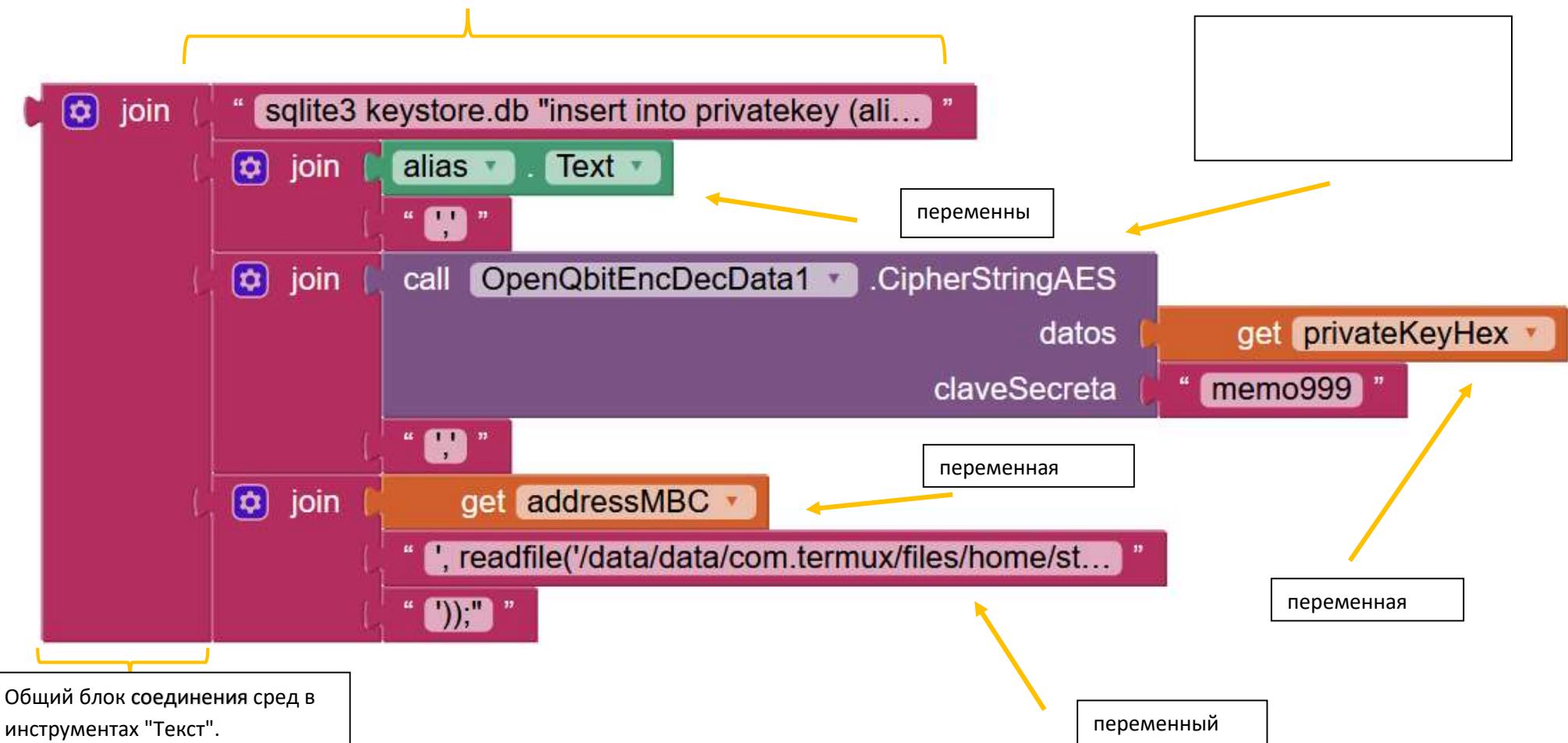


Квантовая серия случайных чисел, генерируемая блоком (ApiGetQRNGInteger), проверяет, как форматировать результат JSON, так как для записи блока (GenerateAddrMiniBlocklyChain) нужна серия чисел, разделенная запятыми ",".

Синтаксис команды очень важен, мы должны ввести следующую команду в правильном формате в среде Blockly.

Значения значений всегда будут переменными, например:

```
sqlite3 keyystore.db "вставить в privatekey (псевдоним, addrHex, addrMBC, addrBin) значения ('memo', 'QWERTY', 'MBC12345',  
readfile('/data/data/com.termux/files/home/storage/dcim/MiniPrivate.key'););".
```



После выполнения блоков мы получим результат в таблице приватных ключей базы данных keystore.db, аналогичный приведенному ниже:

Вводим sqlite обработчик в терминал Termux, открываем базу keystore.db и выполняем предложение:

\$ kv.3

SQLite версия 3.32.2 2020-06-20 15:25:24

Введите ".help" для подсказок по использованию.

Подключен к базе данных переходных процессов в памяти.

Используйте ".open FILENAME" для повторного открытия в постоянной базе данных.

sqlite> .open keystore.db

sqlite> выберите *

```

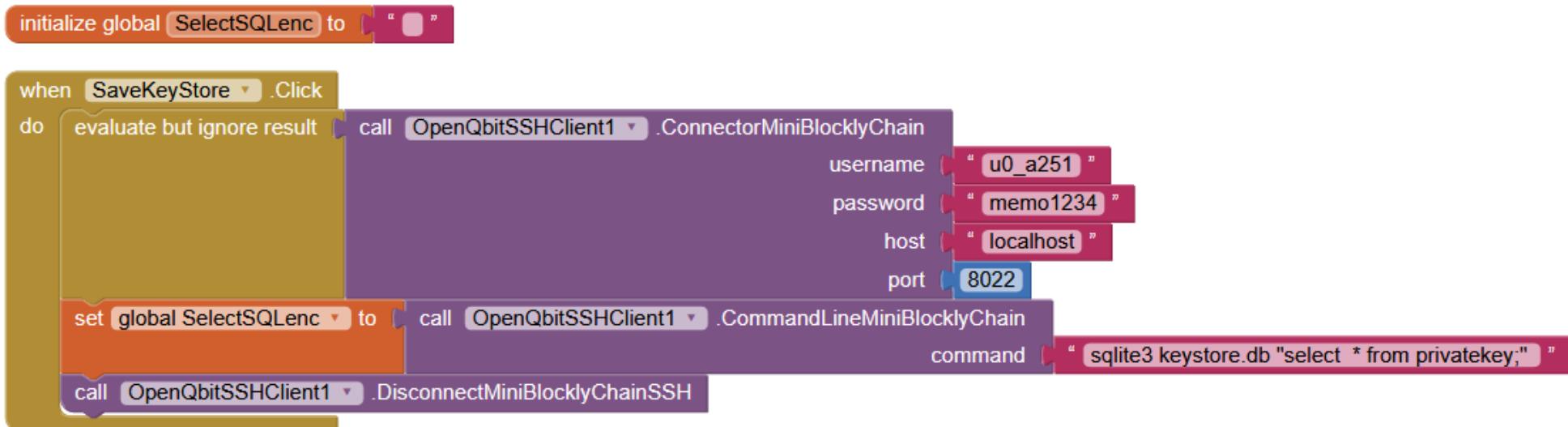
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> select * from privatekey;
1|mexico|07JBBizTcC0Ce6a8Rwe5aTV410L1DKiUQZyPSfJuRbVUZnvIwQJcry4LilBMs5jHavQeMo1jN8rCU87V5XkazYaKR4fswopeTQmI/Q+ipzI=|2eENpFt2HSuGtXNXoeiwfrp5d6
|37Zy5|000
sqlite>
    
```

из личного ключа;

шифрование

адрес addrMBC

КОНСУЛЬТИРОВАНИЕ всех данных в таблице приватных ключей базы данных SQLite **keystore.db**



КОНСУЛЬТИРОВАНИЕ псевдонимов данных в таблице приватных ключей базы данных SQLite **keystore.db**

sqlite3 keystore.db "выберите псевдоним из privatekey;"

Запросить все данные в колонке addrHex, зашифрованные в таблице приватных ключей БД SQLite **keystore.db**.

sqlite3 keystore.db "select addrHex from privatekey;"

CONSULT для получения закрытого ключа addrBin в таблице закрытых ключей базы данных SQLite **keystore.db**

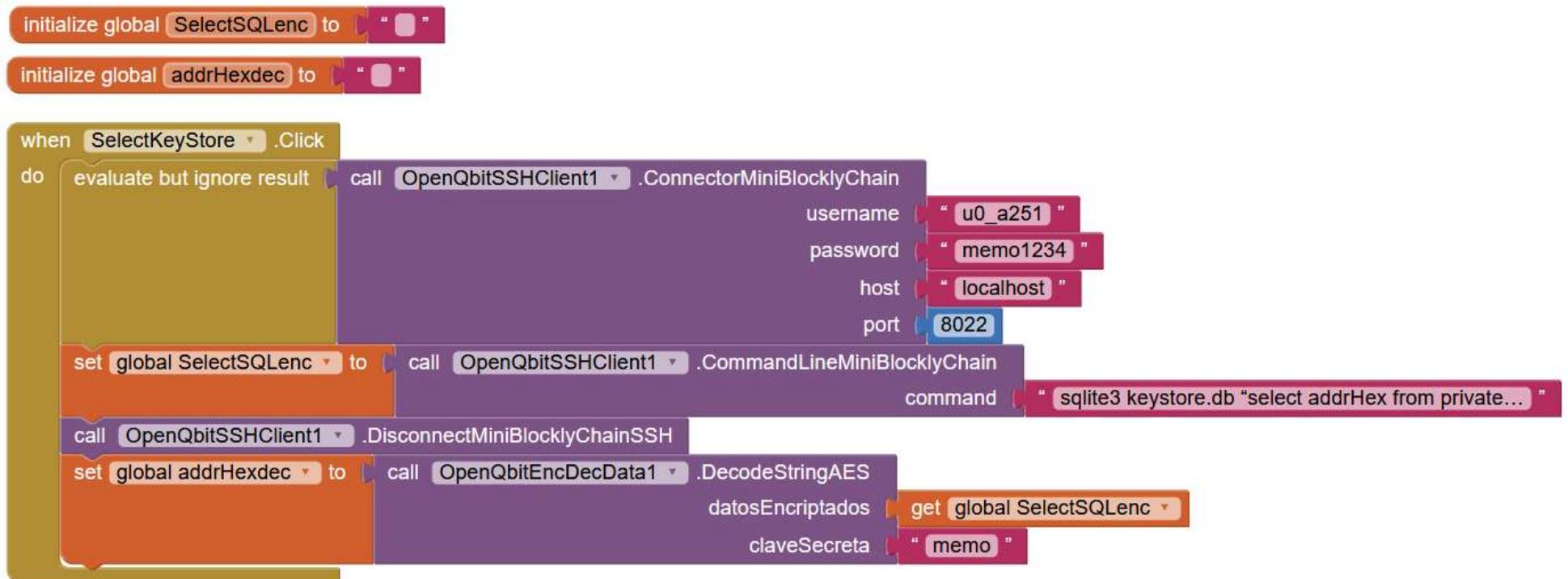
sqlite3 **writefile('PrivateKey.key', addrBin)** ОТ частного ключа Где псевдоним='mexico'; (2)

(1) Этот тип запроса будет основным при обращении к закрытому ключу для выполнения процесса подписания транзакций.

Запрос на расшифровку данных псевдонимом в столбце addrHex таблицы приватных ключей БД SQLite keyystore.db

В этом случае мы будем использовать блок (DecodeStringAES) для декодирования данных, хранящихся в колонке "addrHex".

sqlite3 keyystore.db "select addrHex from privatekey where='mexico';".



Эта консультация дает нам частный ключ в шестнадцатеричном формате, это основная часть любого приема или отправки транзакций. Неоднократно рекомендуется сохранять резервную копию этой базы данных keystore.db.

Дизайн базы данных **publickeys.db** с таблицей **publicaddr**:

\$ kv.3

SQLite версия 3.32.2 2020-06-20 15:25:24

Введите ".help" для подсказок по использованию.

Подключен к базе данных переходных процессов в памяти.

Используйте ".open FILENAME" для повторного открытия в постоянной базе данных.

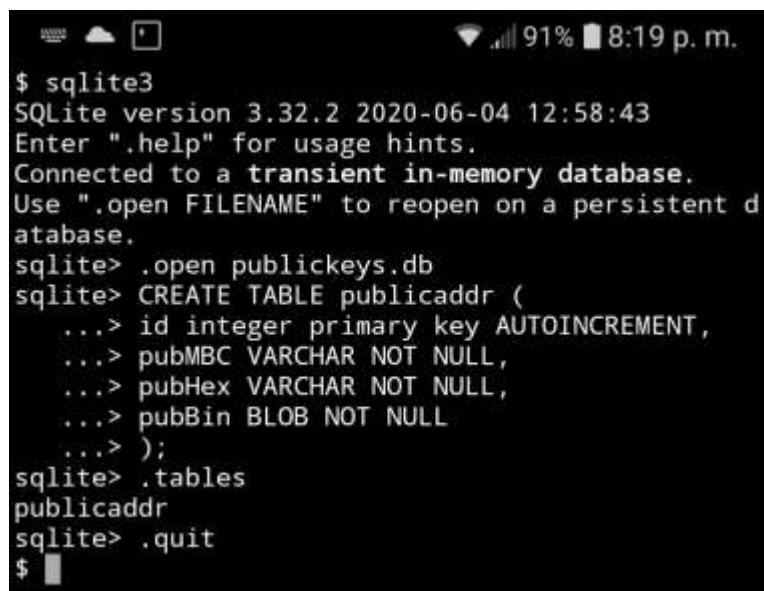
sqlite> .open publickeys.db

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

Далее создание **опубликованной** таблицы показано в виде сегментированного предложения SQL:

```
sqlite> CREATE TABLE publishedaddr (
...> id целочисленный первичный ключ AUTOINCREMENT
...> pubMBC VARCHAR NOT NULL,
...> pubHex VARCHAR NOT NULL,
...> pubBin BLOB NOT NULL
...> );
sqlite> .таблицы
издатель
sqlite> .quit
```



A screenshot of an Android device's terminal window. The title bar shows the time as 8:19 p.m. and battery level at 91%. The terminal output is as follows:

```
$ 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. Приложение "RESTful SQLite GET/POST команды".

Далее показаны различные команды, которые мы можем использовать в Restful SQLite сети резервного копирования. С ними можно ознакомиться, ознакомившись с первоначальной разработкой GITHUB (<https://github.com/olsonpm/sqlite-to-rest>)

Операции CRUD (Создание, считывание, обновление и удаление) Отзывают.

Ниже приведен список операций, доступных через RESTful API в виде псевдо-примеров. Мы будем предполагать базовую таблицу "op.sqlite" и две связанные с ней таблицы "trans" для транзакций и еще один "знак" для адреса источника, адреса назначения и стоимости активов с двумя столбцами id INTEGER PRIMARY KEY.

Как указано в ограничениях, обратите внимание, что методы (DELETE и POST) могут влиять только на один ряд за раз.

ОБТАИН Это обеспечивает наибольшую вариативность. Позже мы увидим все доступные операторы запросов.

Заголовки могут быть указаны чуть ниже URL.

/трансЗапросы
для всех рядов/

/транс?id=1 Где
id=1

транс
диапазон: ряды=0-2Первые
три ряда

/транс
диапазон: ряды=-5 Последние
пять рядов

/транс
диапазон: ряды=0-

Столько строк, сколько может предоставить сервер, что на практике будет меньше maxRange и общего количества строк.

/транс
диапазон: строки=1-

Столько строк, сколько может предоставить сервер, начиная с первой строки.

транс

Заказ

по возрастанию имени

транс

порядок: имя нисходящее по порядку

по имени нисходящее

/транс

порядок: имя внизу, имя внизу

Составлено, но сначала отсортировано по нисходящей фамилии, а в случае ничьей - по восходящей.

/транс?id>1

Куда ты идешь > 1

/trans?id>=2&id<5

Где id >= 2 и id < 5

/транс?name_NOTNULL

Где имя не является нулевым

/trans?name_ISNULL Где

имя нулевое.

**/trans?id!=5&name_LIKE'Spotted%'.
Where id != 5 и имя LIKE "Spotted%" (игнорируйте кавычки)**

**/trans?id>=1&id<10&name_LIKE'Avery%'.
порядок: имя вниз, идентификационный диапазон: строки=2-4**

Внесен для примера.

Получить транзакцию с идентификаторами от 1 до 9 включительно, с именем типа "Avery%", упорядоченным сначала по убыванию имени, а затем по возрастанию идентификатора, получив третью-пятую строку результата. Или на SQL:

DELETE Требуется строка запроса со всеми установленными первичными ключами, равными значению. Это требует максимального однорядного удаления.

/trans?id=1Deletes

trans with id=1

Если вместо этого у сделки был основной ключ, состоящий из идентификатора и имени.

/trans?id=1&name='Avery IPA'.

создание POST

Нельзя передавать строку запроса. Если строка запроса передана, предполагается обновление POST. Все POST-запросы должны передавать тип содержимого заголовка: application / json.

Пожалуйста, обратите внимание, что тело должно содержать все не отменяемые колонки PRIMARY KEY, а не INTEGRATE. В противном случае будет отправлено 400 ответов, указывающих на то, какие поля были потеряны. Нулевые колонки будут нулевыми, а колонки INTEGER PRIMARY KEY будут автоматически увеличены в соответствии со спецификациями sqlite3.

Данные JSON будут указаны чуть ниже URL.

транс

{ "id":1, "name": "Serendipity" } Создает транс с id = 1 и именем = "Serendipity".

транс

{ "id":1 } Создает транс с id = 1 и именем = NULL.

/транс

{ "имя": "Серендибитис" }

Создает сделку с id установленным значением, увеличенным на sqlite3 Спецификации INTEGER PRIMARY KEY.

/транс

{ }

Создает транзакцию с увеличенным id и именем, установленным в NULL.

обновление POST

Она должна содержать строку запроса. Без строки запроса предполагается создание POST. Как и в случае с созданием POST, требуется тип содержимого заголовка: приложение / json.

Строка запроса должна содержать все первичные ключи, чтобы обеспечить обновление только одной строки. Если будут переданы неверные значения, то с помощью клавиш-нарушителей будет возвращено 400.

Тело запроса должно содержать непустой объект и содержать действительные ключи, соответствующие названиям столбцов.

Данные JSON будут указаны чуть ниже URL.

/транс?id=1

{ "id":2 }

Обновите транзакцию с идентификатором 1, установив его в два.

/транс?id=1

" { "имя": MCBza45Rt56cvbgfdR2Swd788kj" }

Обновите транзакцию с идентификатором 1, установив его имя или значение в "MCBza45Rt56cvbgfdR2Swd788kj".

Если бы вместо этого в торговом зале был основной ключ, состоящий из идентификатора и названия.

/trans?id=1&name=MCBza45Rt56cvbgfdR2Swd788kj

{ "имя": "MCB3ofFG5Hj678MNb09vLdfaasX" }

Обновите транзакцию, где id равен единице, а адрес - MCBza45Rt56cvbgfdR2Swd788kj, установив MCB3ofFG5Hj678MNb09vLdfaasX.

Ссылка на

[isSqliteFile](#)

Просто проверьте первые 16 байт файла, чтобы убедиться, что он равен 'sqlite format 3', за которым следует нулевой байт.

[isDirectory](#)

Возвращает результат fs.statsSync с последующим .isDirectory

[isFile](#)

Возвращает результат fs.statsSync с последующим .isFile

операторы GET-консалтинга

Условия запроса должны быть обозначены символами подключения, например, id> 5 & name = MCBza45Rt56cvbgfdR2Swd788kj.

Двоичные операторы (требуется значение после) Человек.

=

!=

>=

<=

>

<

LIKE

LIKE особенный, потому что он должен иметь простые открывающие и закрывающие котировки. В противном случае будет сгенерирована 400 ошибка, показывающая, где анализ не смог быть завершен и что ожидалось. См. примеры операций CRUD RESTful Operations.

Одиночные операторы (должны следовать за названием столбца)

НУЛЬ

НЕ НУЛЬШЕ

Объект конфигурации маршрутизатора

isLadenPlainObject

Назначение данного объекта - предоставить общую конфигурацию для маршрутизатора sqlite. Допускаются следующие свойства:

префикс: isLadenString Стока, передаваемая в опцию сборщика префиксов ко-маршрутизатора. Например, скелетный сервер не указывает префикс, который позволяет пиву API быть удаленным непосредственно из корня http домена: // localhost: 8085 / trans. Если вы установили префикс в '/ api', то вы должны посыпать запросы на http: // localhost: 8085 / api / trans.

allTablesAndViews: табличный объект конфигурации

Настройки, указанные в этом объекте, будут применены ко всем таблицам и представлениям, дополнительно переопределенным свойством tablesAndViews.

tableAndViews: isLadenPlainObject Объект прошлого должен иметь ключи, соответствующие столбцу базы данных или просматривать имена. В противном случае будет выдано дружественное сообщение об ошибке. Значения для каждой таблицы и представления должны быть табличным объектом конфигурации.

Табличный объект конфигурации

isLadenPlainObject Этот объект представляет настройки, которые могут быть установлены для представлений или таблиц. Это позволяет использовать следующие свойства:

maxRange: isPositiveNumber

Приложение по умолчанию: 1000

Это максимальный диапазон, в котором ваш сервер будет разрешать запросы. Если GET-запрос поступает без заголовка диапазона, спецификация предполагает, что вам нужен весь ресурс. Если количество строк, приводящих к GET, больше, чем maxRange, возвращается статус 416 с пользовательским заголовком тах-диапазона. Значение приложения по умолчанию намеренно консервативно в надежде, что авторы установят maxRange в соответствии со своими потребностями.

Обратите внимание, что "Бесконечность" - это действительное положительное число.

флаги: isLadenArray

На данный момент единственным принятым индикатором является строка 'sendContentRangeInHEAD'. При установке, запросы HEAD будут возвращать диапазон доступного контента в форме контент-диапазон: * / <макс-диапазон>. Причина, по которой он настраивается, заключается в том, что вычисление максимального диапазона может быть более трудоемким, чем стоит, в зависимости от нагрузки на сервер и размера ваших таблиц.

Заголовки клиентов

Приложение

порядок: этот заголовок определяется только для GET и может рассматриваться как эквивалент sql ORDER BY. Он должен содержать разделенное запятыми имя столбца, каждый из которых по желанию должен сопровождаться пробелом, а строки - "вверх" или "вниз". В случае отправки неверных значений ордеров, в ответе 400 будет указано, какие именно.

Ответ

Не все из них обязательно индивидуальны, но все их использование выходит за рамки спецификации и поэтому требует уточнения.

GET

тах-диапазон: этот заголовок возвращается, когда количество запрашиваемых строк превышает настроенный maxRange. Обратите внимание, что запрос может не указывать заголовок диапазона, но количество строк, приводящих к этому ресурсу, все равно будет проверено.

диапазон содержания: rfc7233 состояния

Только коды статуса 206 (Частичное содержание) и 416 (Неудовлетворительный диапазон) описывают значение для диапазона Content-Range.

Когда sqlite-to-rest отвечает кодом состояния 200, заголовок диапазона контента отправляется в формате 206 из <начало строки> - <окончание строки> / <счет строки>.

Когда запрос посыпается без заголовка диапазона и полученное количество строк превышает maxRange, возвращается 400 с диапазоном содержимого, установленным в 416 формат * / <количество строк>.

Обратите внимание, что этот заголовок может быть возвращен в ответ HEAD.

accept-order: будет возвращен, если в ордере заголовка запроса указан неверный синтаксис или неправильные имена столбцов. Для получения более подробной информации смотрите раздел HEAD -> Принять-заказ ниже.

25. Приложение "Java Code SQLite-Redis Connector".

Ниже приведен код связи между взаимодействием обеих баз данных SQLite-Redis. В основном, как мы видим, коннектор изменяет формат SQLite на общую строку данных (транзакций) для получения Redis.

Вышеуказанный процесс действует как триггер очереди транзакций для всех подключенных узлов, которые являются возможными кандидатами для обработки текущей очереди транзакций.

Когда база данных Redis получит очередь транзакций по имеющейся у нее конфигурации Master-Slave, она будет распределять информацию в режиме реального времени и в равной степени по всем узлам.

Еще одним фундаментальным моментом является то, что все узлы должны быть синхронизированы в своих часах, это будет сделано, как мы видели ранее, с функциональностью запроса к пулу серверов NTP (Network Time Protocol).

Этот коннектор также выполняет первый уровень проверки безопасности данных, вычисляя дерево Merkle Root всех транзакций.

Базовый код коннектора SQLite-Redis.

```
импорт java.kv.*;
импорт java.util.*
импорт java.util.массивов;
импорт java.util.list;
импорт java.util.массива;
import java.security.*;
```

```

импорт java.security.MessageDigest;
импорт redis.clients.jedis.Jedis;
класс RediSqlite{
    публичный Стинг Предыдущий Хэш;
    общественный Стинг Меркл-Рут;
    публичный статический ArrayList<String> транзакции = новый ArrayList<String>();
    public static ArrayList<String> treeLayer = новый ArrayList<String>();
    публичная статическая строка getMerkleRoot() {
        int count = transactions.size();
        int item = 1;
        Нечетный int = 0;
        ArrayList<String> previousTreeLayer = транзакции;
        пока(счет > 1) {
            treeLayer = новый ArrayList<String>();
            for(int i=1; i <= count ; i=i+2) {

                если (!esPar(count) && i == count) odd = 1;
                treeLayer.add(applySha256(previousTreeLayer.get(i-item)
                + previousTreeLayer.get(i-impar)));

            }
            пункт = 1;
            Нечетный = 0;
            count = treeLayer.size();
            previousTreeLayer = treeLayer;
        }
        Стока merkleRoot = (treeLayer.size() == 1) ? treeLayer.get(0) : "";
        вернуть merkleRoot;
    }
    //Дефинирует ли Меркл Три четный или нечетный/
    статический логический enPar(int номер){
        если (нумеро%2==0) вернуть true; иначе вернуть false;
    }

    публичная статическая строка applicationSha256(Строковый вход){
        пытаюсь
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        //Прилагает Sha256 к нашему входу,
        byte[] hash = digest.digest(input.getBytes("UTF-8"));
    }
}

```

```

StringBuffer hexString = new StringBuffer(); // Это будет содержать хэш как
шестнадцатеричный
для (int i = 0; i < длина хэша; i++) {
    Страна hex = Integer.toHexString(0xff & hash[i]);
    if(hex.length() == 1) hexString.append('0');
    hexString.append(hex);
}
вернуть hexString.toString();
}
catch(Exception e) {
    бросить новый RuntimeException(e);
}
}

public static void main(String args[]){
    попытка
        Соединение     con=DriverManager.getConnection("jdbc:sqlite:C://memo/sqlite-tools-
win32-x86-3310100/trans.sqlite3");
        //Подключение к серверу Redis на локальном хосте.
        jedis = новый Jedis ("местный хозяин");
        jedis.auth("memo1234");
        Statement stmt=con.createStatement();
        Страна sql = "ВЫБРАТИ * С пивоварни";
        ResultSet rs=stmt.executeQuery(sql);
        в то время как(rs.next()) {
            transactions.add(rs.getString(2));
            jedis.set("LATAM:"+String.valueOf(rs.getInt(1)), "["+""""+rs.getString(2)+""""+","+""""+rs.getStri
ng(3)+"","",+""""+rs.getString(4)+""""+")));
        }
        jedis.set("LATAM:merkleroot", getMerkleRoot());
        System.out.println("Количество элементов ArrayList:: "+treeLayer.size());
        c .close();
    }catch(Exception e){ System.out.println(e);}
}

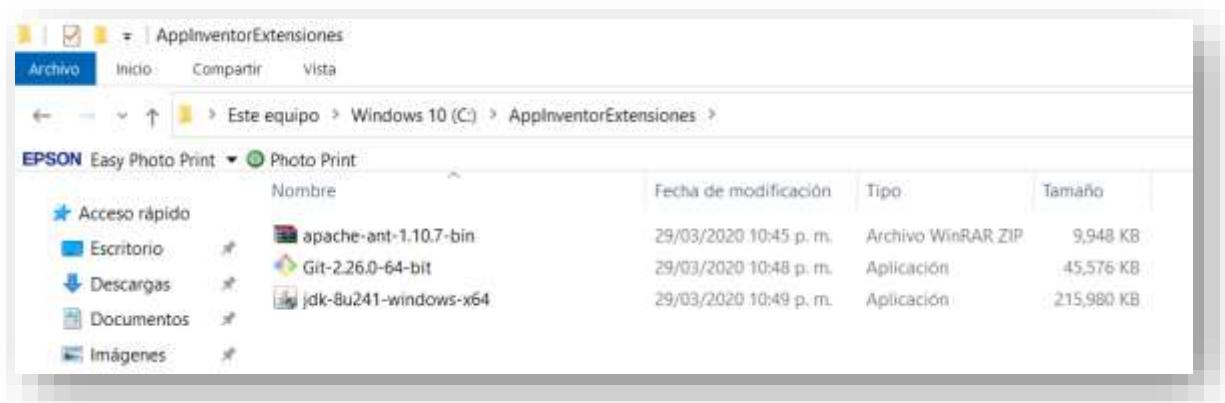
}

```

26. Приложение "Mini BlocklyChain для разработчиков".

В этом приложении мы рассмотрим конфигурацию, установку и основное использование того, как генерировать внешние модули на основе языка программирования Java для среды Blockly, а также сможем создавать специализированные модули для каждого бизнес-кейса и вставлять функциональные возможности в Mini BlocklyChain System или добавлять другие функциональные возможности к нашему мобильному приложению.

Мы создали в нашей системе Windows каталог под названием "AppInventorExtensions", в который будут загружены следующие общедоступные программные пакеты.



1.- Мы собираемся загрузить последнюю версию JDK (Java Development Kit).

пример: jdk-8u251-windows-x64.exe (211 МБ)

<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>



2.- Библиотека Apache Ant, использующая JAVA для сборки приложений, <http://ant.apache.org/bindownload.cgi>, в моем случае я скачал Ant 1.10.8 (Двоичные дистрибутивы) (apache-ant-1.10.8-bin.zip). Могут быть и более продвинутые версии.

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.- Мы установили Git Bash с вашего сайта <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.- Распаковка "Apache Ant". Когда вы закончите распаковку, вы можете сделать это, например, в двойной папке:

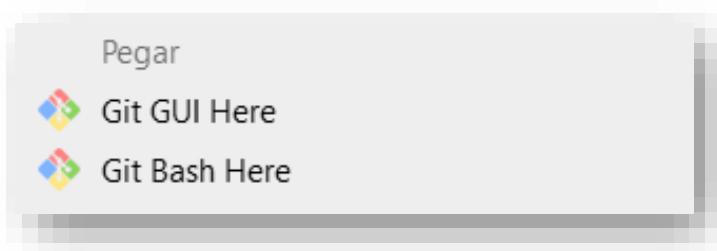
C:\AppInventorExtensions\apache-ant-1.10.8-bin\apache-ant-1.10.8-bin

- Поменяйте на C:\Apache-ant-1.10.8-bin.

Este equipo > Windows 10 (C:) > AppInventorExtensions > apache-ant-1.10.7 >					
Print	Photo Print	Nombre	Fecha de modificación	Tipo	Tamaño
o		bin	01/09/2019 11:43 a. m.	Carpeta de archivos	
o		etc	01/09/2019 11:43 a. m.	Carpeta de archivos	
o		lib	01/09/2019 11:43 a. m.	Carpeta de archivos	
is	#	manual	01/09/2019 11:43 a. m.	Carpeta de archivos	
o		CONTRIBUTORS	01/09/2019 11:43 a. m.	Archivo	7 KB
o		contributors	01/09/2019 11:43 a. m.	Documento XML	33 KB
o		fetch	01/09/2019 11:43 a. m.	Documento XML	14 KB
o		get-m2	01/09/2019 11:43 a. m.	Documento XML	5 KB
o		INSTALL	01/09/2019 11:43 a. m.	Archivo	1 KB
o		KEYS	01/09/2019 11:43 a. m.	Archivo	94 KB
o		LICENSE	01/09/2019 11:43 a. m.	Archivo	15 KB
o		NOTICE	01/09/2019 11:43 a. m.	Archivo	1 KB
o		patch	01/09/2019 11:43 a. m.	Documento XML	2 KB
o		README	01/09/2019 11:43 a. m.	Archivo	5 KB
is		WHATSNEW	01/09/2019 11:43 a. m.	Archivo	250 KB

5.- Мы установили Git Bash. Мы оставили все по умолчанию в установке.

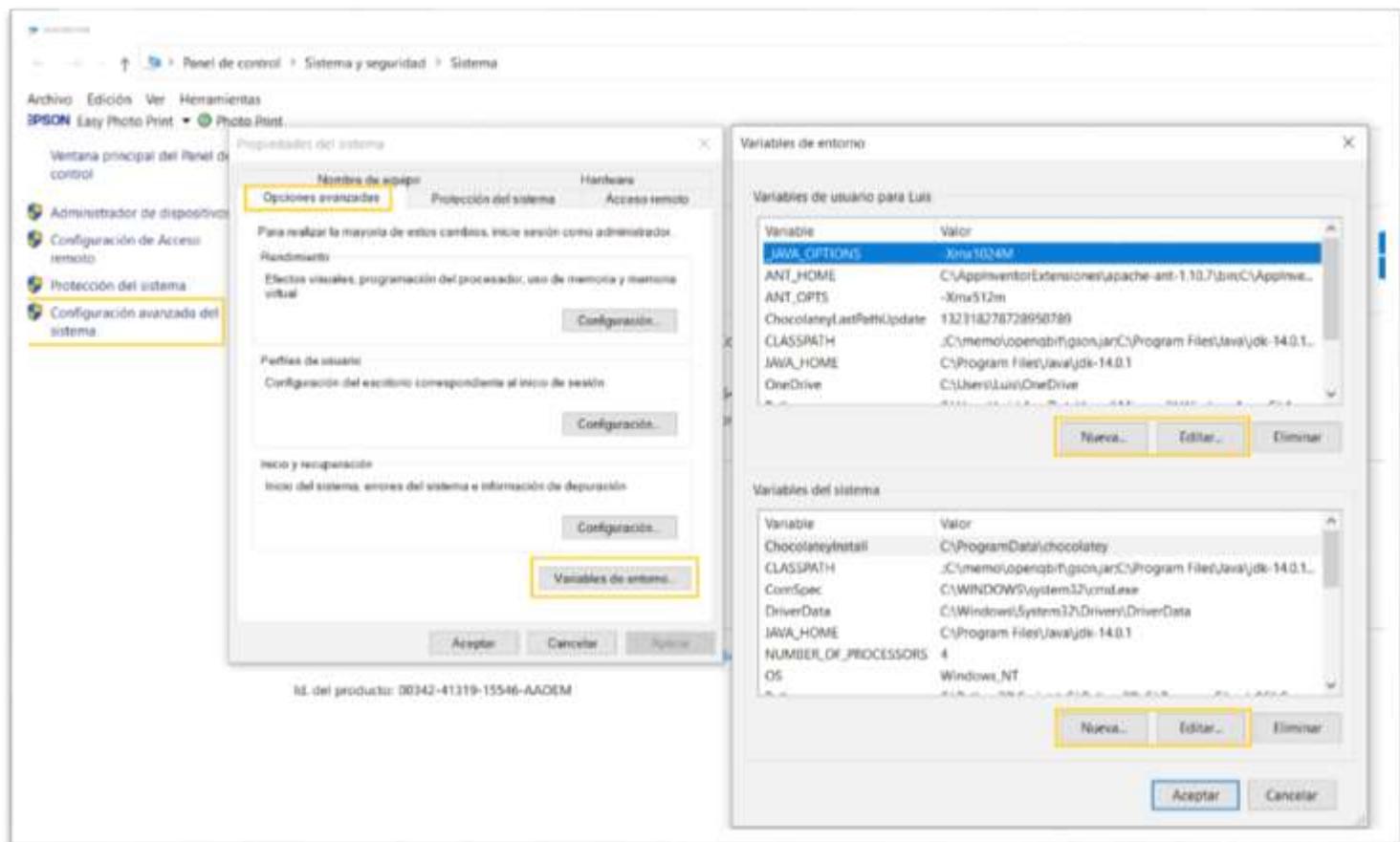
- Позже мы увидим, что у него есть ссылка в кнопке Пуск Windows, нажав левую кнопку в браузере файлов.



6.- Мы установили комплект для разработки Java SE. В зависимости от версии Windows вам, вероятно, придется перезагрузить компьютер.

Переменные системного окружения Windows. Мы создадим переменные окружения. Для этого мы сделаем это:

Панель управления -> Система -> Дополнительные системные настройки -> Дополнительные параметры -> Переменные среды.



В зависимости от того, хотим ли мы поставить новую переменную или отредактировать существующую, мы нажимаем кнопку "Создать..." или "Редактировать...".

Чтобы добавить адреса к уже установленным, мы разделяем их точкой с запятой;

В разделе "Переменные пользователя" для Джона мы установили эти новые...

JAVA_OPTIONS мы ставим Вам Value -Xmx1024m

ANT_HOME мы поместили значение C:\AppInventorExtensions\apache-ant-1.10.8-bin [то есть папку, в которой мы распаковали apache-ant].

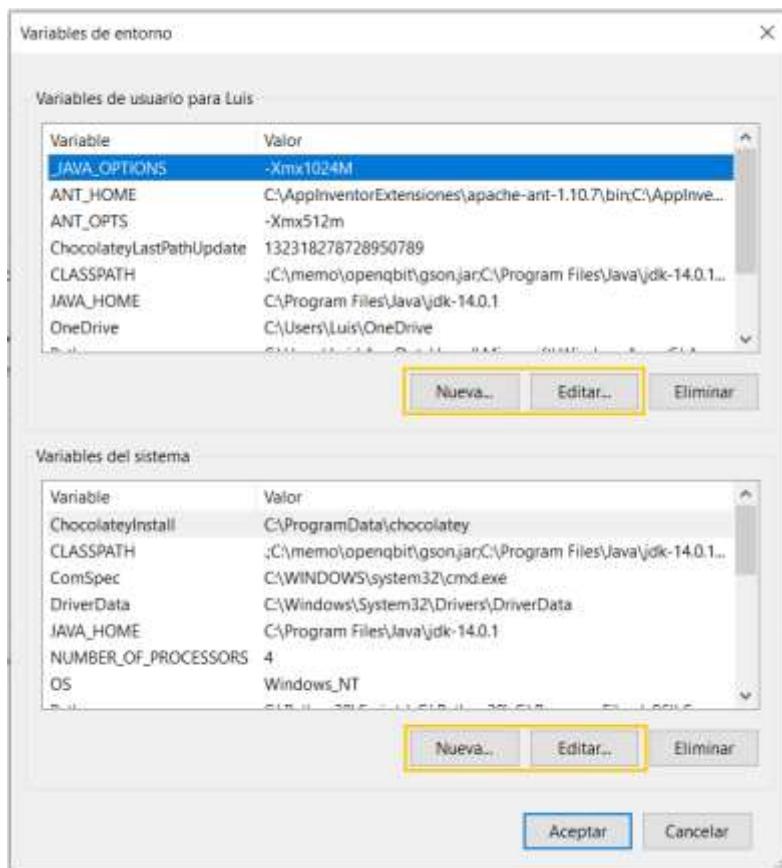
ANT_OPTS мы ставим Value -Xmx256M

JAVA_HOME имеет значение C:³³³³Файлы программы³³³³jdk1.8.0_131 [Если оно имеет другое значение, измените его. Обратите внимание, что это jdk НЕ jdr].

CLASSPATH мы ставим значение %ANT_HOME%\lib;%JAVA_HOME%\lib.

В PATH мы добавили ;%ANT_HOME%\bin;%JAVA_HOME%\bin [Обратите внимание, что; начинается с точки с запятой; добавлять к тем, которые уже есть].

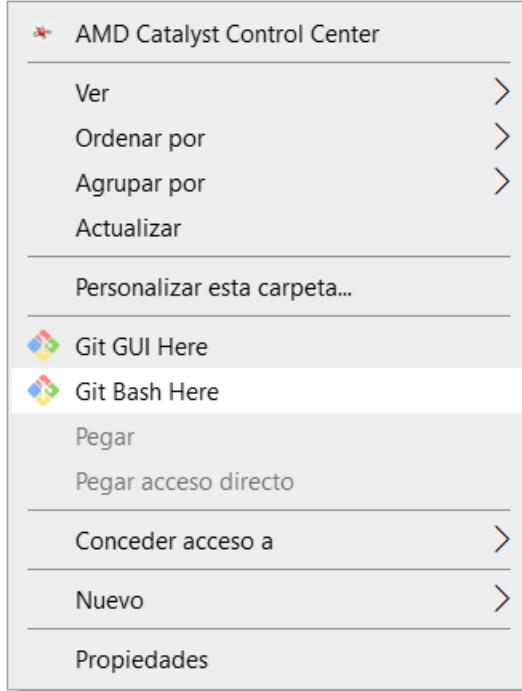
ПРИМЕЧАНИЕ: Переменные отделены друг от друга точкой с запятой: переменная-один; переменная-n; переменная-n+1.



Создание клона App Inventor в нашем компьютере.

Мы создадим "клонированную" копию App Inventor на нашем сервере (ПК), загрузим ее прямо из Интернета и создадим эту копию.

Для этого воспользуемся приложением **Git Bash** и нажмем на него, чтобы открыть терминал.



Мы запускаем команду на терминале Git Bash:

\$ git-клон <https://github.com/mit-cml/appinventor-sources.git>

```
git clone https://github.com/mit-cml/appinventor-sources.git
Cloning into 'appinventor-sources...'...
remote: Counting objects: 43191, done.
remote: Total 43191 (delta 0), reused 0 (delta 0), pack-reused 43190
receiving objects: 300K (43191/43191), 533.30 MiB / 494.00 KiB/s, done.
resolving deltas: 300K (2300/2300), done.
Checking out files: 100% (3058/3058), done.
```

Сайт, на котором находится репозиторий:

<https://github.com/mit-cml/appinventor-sources/>

Она создаст папку с именем "Appinventor-источники" с именем "App Inventor-источник".

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

Мы создали следующий каталог в пути C: Users - Apppinventor-sourcesv-babkup - Apppinventor-components -rc-openqbit

Внутри мы скопировали нашу следующую тестовую программу под названием OpenQbitQRNGch.java

Nombre	Fecha de modificación	Tipo	Tamaño
OpenQbitQRNGch	11/06/2020 01:39 a. m.	Archivo JAVA	5 KB

Создание расширения.

ОТМЕНИТЬ КАПЫ И МИНУСКУЛЬТЫ, это не одно и то же Привет и Привет.

Не делай никаких акцентов. Мы готовы сделать наше первое расширение. Это будет генератор квантовых случайных чисел.

Мы используем Text Editor, Notepad++, мы создаем файл под названием OpenQbitQRNGch.java, который генерирует случайные квантовые числа со следующим кодом:

```
// Это расширение используется для вывода Quantum Random Number Generator
QRNG Switzerland API.

пакет com.openqbit. OpenQbitQRNGch;
импорт com.google.appinventor.components.annotations.DesignerComponent;
импорт com.google.appinventor.components.annotations.DesignerProperty;
импорт com.google.appinventor.components.annotations.PropertyCategory;
импорт com.google.appinventor.components.annotations.SimpleEvent;
импорт com.google.appinventor.components.annotations.SimpleFunction;
импорт com.google.appinventor.components.annotations.SimpleObject;
импорт com.google.appinventor.components.annotations.SimpleProperty;
импорт com.google.appinventor.components.common.ComponentCategory;
импорт com.google.appinventor.components.common.PropertyTypeConstants;
импорт com.google.appinventor.components.runtime.util.MediaUtil;
импорт com.google.appinventor.components.runtime.*;
импорт java.io.*;

@DesignerComponent(version = OpenQbitQRNGch.VERSION,
    описание = "Генератор квантовых случайных чисел API. Квантовые
случайные числа как сервис, QRNGaaS, веб-интерфейс API для генератора
квантовых случайных чисел Quantis, разработанный швейцарской компанией -
" + "Guillermo Vidal - OpenQbit.com",
    Категория = КомпонентКатегория.ВЫПОЛНЕНИЕ
    невидимый = правда,
    iconName = "http://www.pinntar.com/logoQbit.png")
@SimpleObject(external = true)

публичный класс OpenQbitQRNGch расширяет AndroidNonvisibleComponent
реализации Component {

    общедоступная статическая конечная информация в ВЕРСИИ = 1;
    публичная статическая конечная строка DEFAULT_TEXT_1 = "";
    частный контейнер ComponentContainer;
    private String text_1 = "";

    публичный OpenQbitQRNGch (контейнер ComponentContainer) {
        super(container.$form());
        этот.контейнер = контейнер;
    }

    // Создание собственности.
    //@DesignerProperty(editorType =
    PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue =
    OpenQbitQRNGch.DEFAULT_TEXT_1 + "")
    //@SimpleProperty(description = "API получает генератор квантовых
    случайных чисел, случайное число от 0 до 1. Введите переменную *quantity*
    чисел для генерации").

    @SimpleFunction(description = "API получает генератор квантовых
    случайных чисел, случайное число от 0 до 1. Введите переменную целое
    число *количество* генерируемых чисел").
}
```

```
public String APIGetQRNGdecimal(String qty) выбрасывает Исключение {
    Стока url = "http://random.openqu.org/api/rand?size=" + qty;
    Команда String[] = {"curl", url };

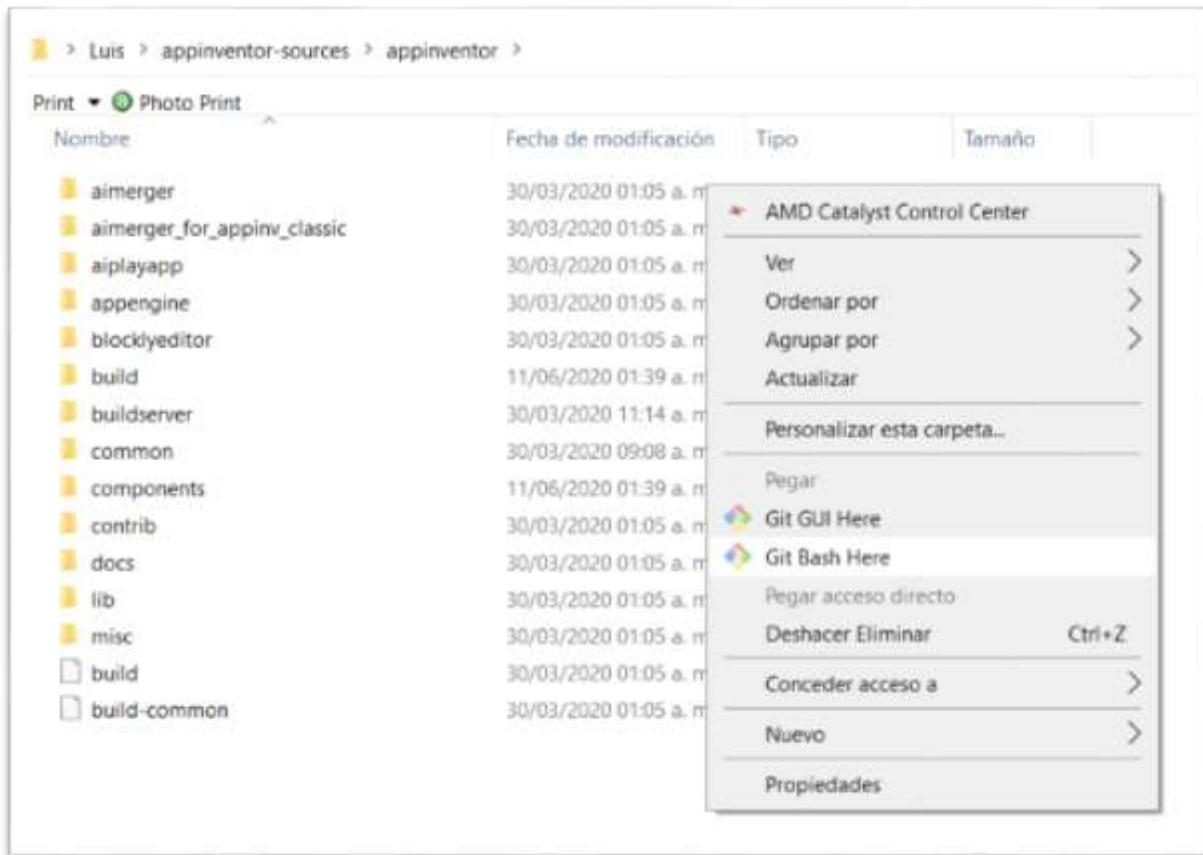
    ProcessBuilder process = новый ProcessBuilder(команда);
    Процесс стр;
    p = process.start();
        BufferedReader reader = новый BufferedReader(new
InputStreamReader(p.getInputStream()));
        StringBuilder builder = new StringBuilder();
        Стока строки = ноль;
        пока ((line = reader.readLine()) != null) {
            builder.append(line);
            builder.append(System.getProperty("разделитель
строк"));
        }
    Результат строки = builder.toString();
    //система.out.print(result);
    Результат возврата;

}
@SimpleFunction(description = "API Get Quantum Random Number Generator,
random number between a range min to max numbers"). Введите переменную
целое число *количество* чисел для генерации диапазона от минимального
*мин* числа до максимального *макс* числа").
public String APIGetQRNGinteger(Стока qty, строка min, строка max)
выбрасывает Исключение {
    Стока url = "http://random.openqu.org/api/randint?size=" + "qty" +
"&min=" + "min" + "&max=" + макс.
    Команда String[] = {"curl", url };

    ProcessBuilder process = новый ProcessBuilder(команда);
    Процесс стр;
    p = process.start();
        BufferedReader reader = новый BufferedReader(new
InputStreamReader(p.getInputStream()));
        StringBuilder builder = new StringBuilder();
        Стока строки = ноль;
        пока ((line = reader.readLine()) != null) {
            builder.append(line);
            builder.append(System.getProperty("разделитель
строк"));
        }
    Результат строки = builder.toString();
    //система.out.print(result);
    Результат возврата;

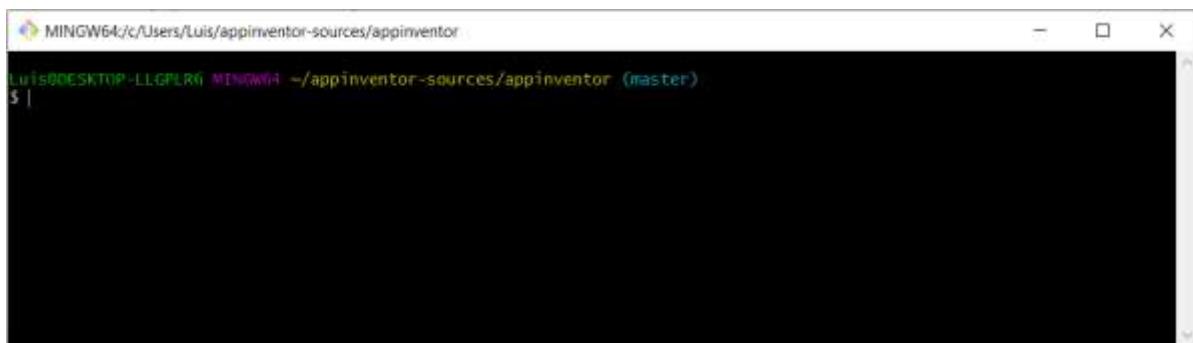
}
}
```

Мы выполняем **Git Bash**, позиционируя себя по следующему пути: **C: Luis-appinventor-sources-appinventor**. В этом каталоге мы нажимаем левую кнопку мыши и выбираем терминал **Git Bash**.



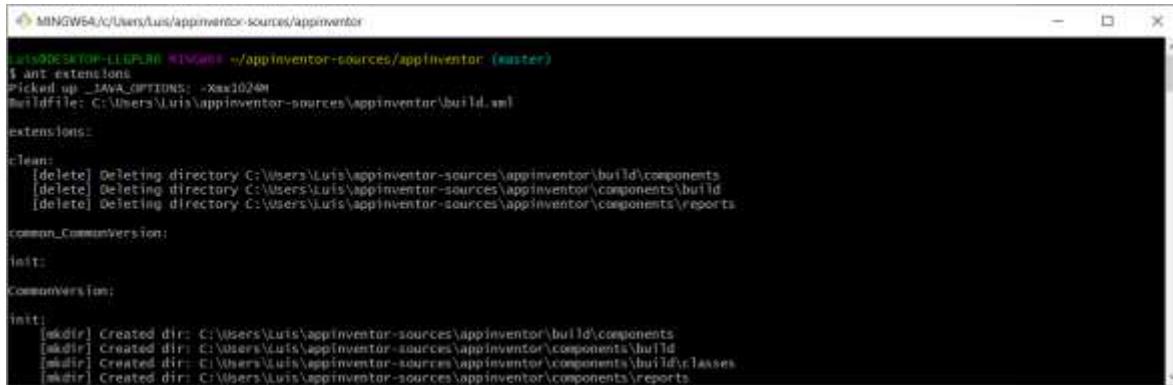
Терминал Git bash будет расположен в следующем каталоге:

C:\Users\Luis\appinventor-sources\appinventor (мастер)



В терминале Git Bash выполните следующую команду:

муравьиные продления



```
MINGW64/C:/Users/Luis/appinventor-sources/appinventor
Luis@DESKTOP-LLGPX90 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

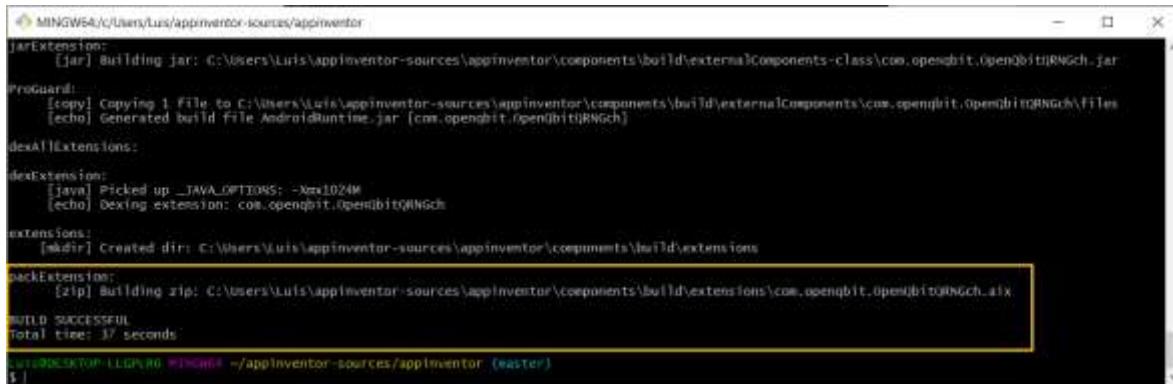
common_ConnectVersion:
init:
commonVersion:
init:
[mkdir] Created dir: C:/Users/Luis/appinventor-sources/appinventor/build/components
[mkdir] Created dir: C:/Users/Luis/appinventor-sources/appinventor/components/build
[mkdir] Created dir: C:/Users/Luis/appinventor-sources/appinventor/components/build/classes
[mkdir] Created dir: C:/Users/Luis/appinventor-sources/appinventor/components/reports
```

Если все пройдет хорошо, мы его получим: СТРОИТЬ УСПЕШНО.

Наше расширение было создано в...

C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions

ПРИМЕЧАНИЕ: содержимое папки расширений удаляется и воссоздается каждый раз, когда мы делаем расширение муравейника.



```
MINGW64/C:/Users/Luis/appinventor-sources/appinventor
JarException:
[jar] Building jar: C:/Users/Luis/appinventor-sources/appinventor/components/build/externalComponents-class/com.openqbit.OpenQbitQRNGch.jar
Preguard:
[copy] Copying 1 File to C:/Users/Luis/appinventor-sources/appinventor/components/build/externalComponents/com.openqbit.OpenQbitQRNGch/Files
[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
packageExtension:
[zip] Building zip: C:/Users/Luis/appinventor-sources/appinventor/components/build/extensions/com.openqbit.OpenQbitQRNGch.aix
BUILD SUCCESSFUL
Total time: 37 seconds
Luis@DESKTOP-LLGPX90 MINGW64 ~/appinventor-sources/appinventor (master)
$
```

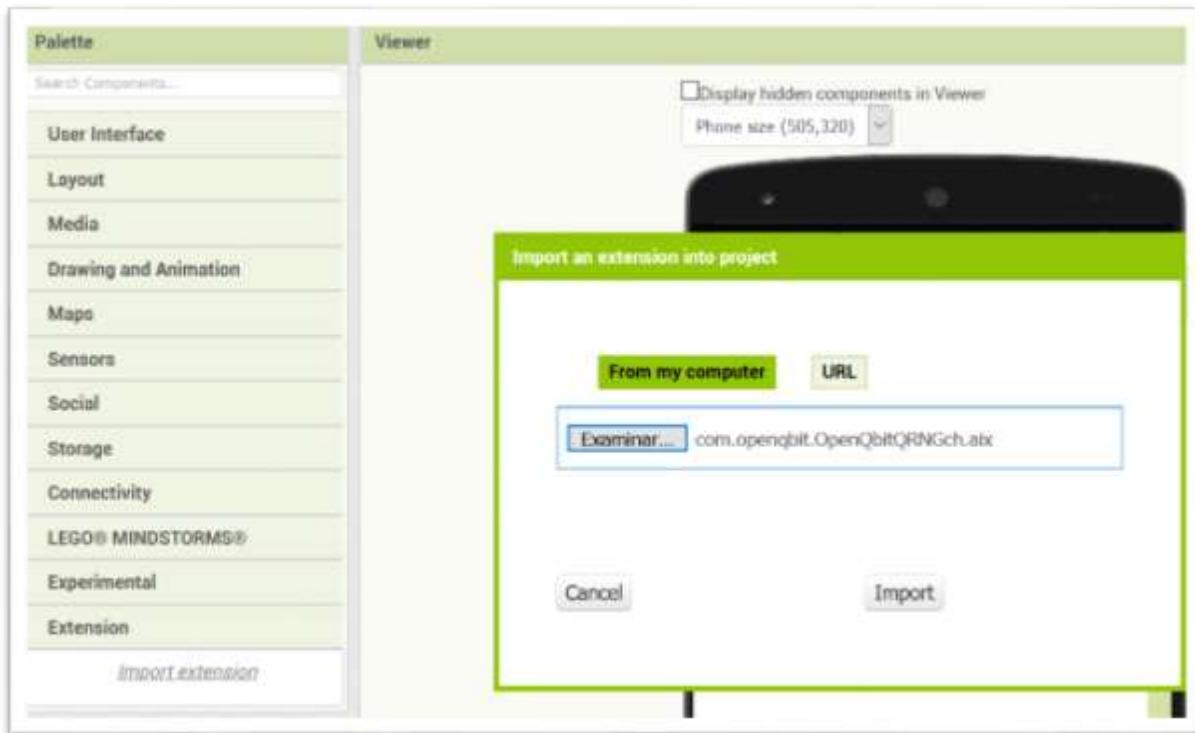
Пойдем в эту папку:

C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions

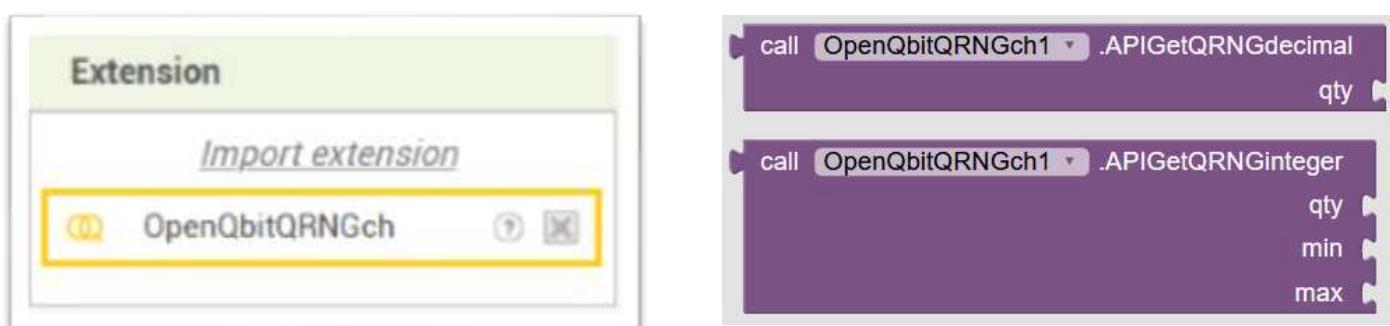
и скопируйте файл com.openqbit.OpenQbitQRNGch.aix, этот файл является нашим расширением.

Имея учетную запись в App Inventor или другой системе Blockly, мы вошли, чтобы добавить новое расширение и протестировать его.

Перейдите в нижнюю часть, где находится опция "Расширение" и нажмите на кнопку "Импорт расширения":



Мы нажали на кнопку "Импорт". Теперь у нас есть блоки (APIGetQRNGdecimal) и (APIGetQRNGinteger), которые будут использоваться:



У нас уже есть первое созданное и функциональное расширение.

27. Приложение "Интеллектуальные контракты с блочным кодом".

BlocklyCodes - это программы, созданные на языке java. Расширение для создания такого типа программ, обычно называемое "Умные контракты" - это способ обработки соглашений между пользователями (компаниями или людьми).

Данный блок (**BlocklyCode**), реализован в программе, параметры которой уже установлены и которая в зависимости от типа договора может быть выполнена системой Mini BlocklyChain в автоматическом режиме при выполнении помещений, которые управляют "умным договором". Параметры, которые следует учитывать при рассмотрении предложенных или входных параметров "входов", являются следующими

Срок годности

Указанная дата

Время истечения

Указанное время

Связанный актив

Всего основных фондов

Переменные активы

Участники договора

Подтвержденные данные (Строка)

Подтвержденные данные (ФИО)

Переменное событие

Исправленное событие

Проверка документа(ов)

Проверка струн

Подпись действительна

Определенный интервал (Целое число)

Десятичный интервал

Установленный минимум

Установленный максимум

Перед началом использования блока (**BlocklyCode**) необходимо сначала установить систему, которая будет выполнять исполнение "Smart Contracts", это делается путем установки в терминале Termux OpenJDK и OpenJRE.

OpenJDK (Open Java Development Kit). - Это инструмент для разработки программ на java, он содержит библиотеки, компилятор и JVM (Java Virtual Machine).

OpenJRE (Open Java Runtime Environment). - Это инструмент только для запуска java-программ. Он содержит JVM (Java Virtual Machine).

Продолжаем установку OpenJRE и OpenJDK в терминал Termux на узлах, образующих систему Mini BlocklyChain. Мы установили комнаты

\$ подходящая установка - и wget



Мы скачали 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_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 18.90M 1.06MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 19.10M 1.10MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 19.30M 1.09MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 19.52M 1.08MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 19.76M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 19.96M 1.06MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 20.21M 1.10MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 20.44M 1.11MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 20.63M 1.09MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 20.87M 1.09MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 21.07M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 21.32M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 21.57M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 21.80M 1.06MB/s
openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb | 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]

$
```

Мы скачали 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://ia802807.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://ia802807.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 ia802807.us.archive.org... 207.241.232.117
Connecting to ia802807.us.archive.org|207.241.232.117|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 34976 (34R) [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_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb | 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]

$
```

Мы осуществляем установку с терминала Termux OpenJDK и OpenJRE:

```
$ apt установка -и ./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
) ...
$ █
```

С тех пор как мы закончили установку окружения OpenJDK и OpenJRE. Эти установки содержат JVM (Java Virtual Machine), в которой будет запущена среда BlocklyCode.

Теперь мы установим редактор для создания файла онлайн, мы будем использовать редактор под названием **vi** выполнить следующую команду:

```
$ apt install vim
```

Теперь попробуем скомпилировать тестовый файл и запустить его. Мы создаем в терминале Termux с помощью редактора **vi** и пишем java-код "Hello World", а затем сохраняем его в файле HelloWorld.java.

```
общественный класс HelloWorld

public static void main(String[] args) {
    // Печатает "Привет, мир" в окно терминала.
    System.out.println ("Привет, мир");
}

}
```

Затем мы запускаем следующую команду в терминале Termux для компиляции, а затем выполняем программу:

```
$ javac HelloWorld.java (После запуска создается байткод файла HelloWorld.class)
```

```
$ java HelloWorld (После запуска и печати объявления, Hello World)
```



Используя блок (**BlocklyCode**) этот блок находится в расширении (**ConnectorSSHClient**).



В предыдущем блоке можно посмотреть, как будет выполнен файл `HelloWorld.class`, который уже был скомпилирован и размещен в каталоге базы пользователей.

Это блок, в котором можно выполнить уже скомпилированную на java программу, которая широко известна в среде разработки как файл байткода в его двоичном виде.

Этот тип файла готов к выполнению с помощью виртуальной машины Java (JVM).

Есть два способа создать файл байткода или с расширением .CLASS, пользователь может создать его на своем компьютере с комфортом или его можно сделать и в мобильном телефоне, помните, что мы уже установили среду для компиляции JDK (Java Development Kit) раньше, хотя это будет менее эффективно, потому что ограничения клавиатуры будут считаться, также в случае выбора среды Android придется учитывать, что библиотеки ограничены OpenJDK, которая была установлена.

Входные параметры открыты в `<inputsValue>`, а остальные фиксированные - в `activeValue`, `fromaddrMBC` и `toAddrMBC`.

В случае проведения тестов они могут быть оставлены пустыми без содержимого, а без параметров будет выполняться только файл байткода.

Предыдущий блок похож на следующую выполняемую командную строку:

```
$ java HelloWorld
```

Программы, разработанные для **BlocklyCode**, будут подчиняться каждой конкретной среде проектирования и могут управляться и выполняться рутинно с помощью агента "cron" и совместно с синхронизатором.

28. Приложение "OpenQbit Quantum Computing".

Как работают квантовые вычисления? ⁽²⁾

Цифровое преобразование приводит к изменениям в мире быстрее, чем когда-либо. Вы бы поверили, что цифровая эра вот-вот закончится? **Цифровая грамотность** уже была определена как область, в которой открытые знания и доступные возможности для изучения технологий являются настущей необходимостью для устранения пробелов в социальном и экономическом развитии. Обучение на основе ключевых концепций цифровой эры станет еще более критичным с неизбежным приходом еще одной новой технологической волны, способной трансформировать существующие модели с поразительной скоростью и мощью: **квантовые технологии**.

В этой статье мы сравниваем основные понятия традиционных и квантовых вычислений, а также начинаем изучать их применение в других смежных областях.

Что такое квантовые технологии?

На протяжении всей истории люди разрабатывали технологии, понимая, как работает природа с помощью науки. Между 1900 и 1930 годами, изучение некоторых физических явлений, которые еще не были хорошо изучены, привело к созданию новой физической теории, **Квантовая механика**. Эта теория описывает и объясняет функционирование микроскопического мира, естественную среду обитания молекул, атомов или электронов. Благодаря этой теории удалось не только объяснить эти явления, но и понять, что субатомная реальность работает совершенно интуитивно, почти магически, и что в микроскопическом мире происходят события, которые не происходят в макроскопическом мире.

Эти квантовые **свойства** включают в себя квантовое суперпозиционирование, квантовую связь и квантовую телепортацию.

- **Квантовая суперпозиция** описывает, как частица может находиться в разных состояниях одновременно.
- **Квантовая связь** описывает, как две частицы, находящиеся так далеко друг от друга, могут быть соотнесены таким образом, что при взаимодействии с одной из них другая осознает это.
- **Квантовая телепортация** использует квантовую связь для передачи информации из одного места в другое в пространстве без необходимости путешествовать через нее.

Квантовые технологии основаны на этих квантовых свойствах субатомной природы.

В этом случае сегодня понимание микроскопического мира через Квантовую механику позволяет нам изобретать и проектировать технологии, способные улучшить жизнь людей. Существует много и очень разных технологий, использующих квантовые явления, и некоторые из них, такие как лазеры или магнитно-резонансная томография (МРТ), существуют у нас уже более полувека. Однако в настоящее время мы являемся свидетелями технологической революции в таких областях, как квантовые вычисления, квантовая информация, квантовое моделирование, квантовая оптика, квантовая метрология, квантовые часы или квантовые датчики.

Что такое квантовые вычисления? Во-первых, вы должны понять классические

Caracter	Bits
7	111
A	01000001
\$	00100100
:)	0011101000101001

вычисления.

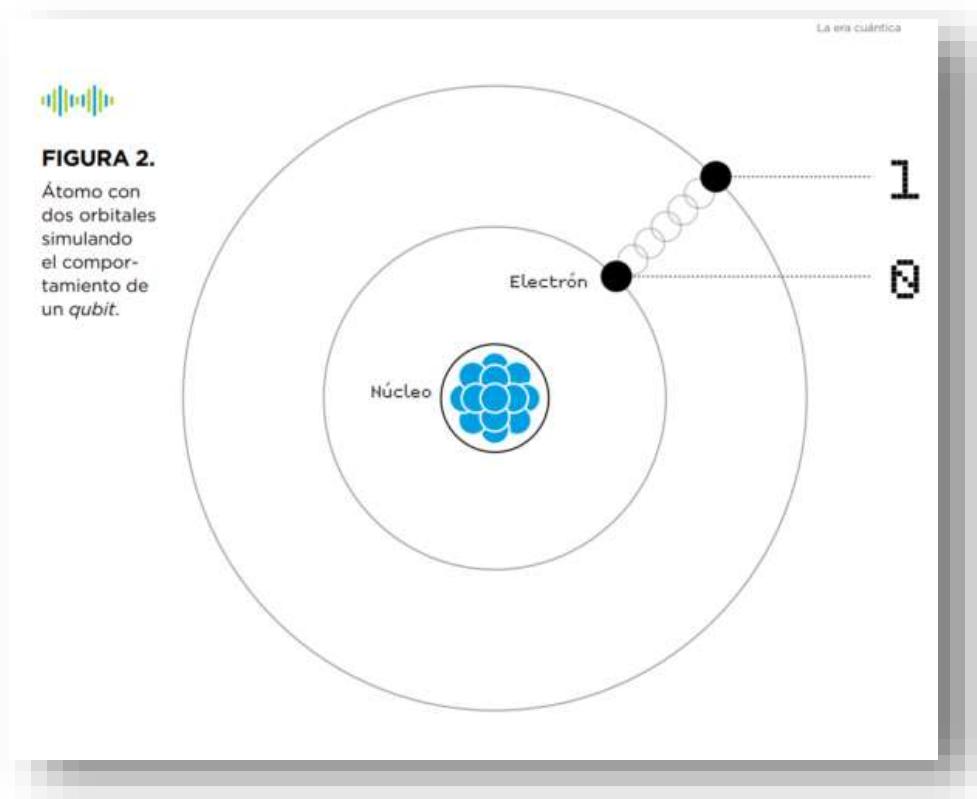
Чтобы понять, как работают квантовые компьютеры, удобно сначала объяснить, как работают компьютеры, которыми мы пользуемся каждый день и которые мы будем называть в этом документе цифровыми или классическими компьютерами. В них, как и в других электронных устройствах, таких как планшетные компьютеры или мобильные телефоны, биты используются в качестве основных блоков памяти. Это означает, что программы и приложения кодируются в битах, то есть в двоичном языке нулей и единиц. Каждый раз, когда мы взаимодействуем с любым из этих устройств, например, нажимая клавишу на клавиатуре, внутри компьютера создаются, уничтожаются и/или модифицируются строки нулей и единиц.

Интересный вопрос - что это за нули и физические нули внутри компьютера? Ноль и одно состояние соответствуют электрическому току, который циркулирует или не циркулирует через микроскопические части, называемые транзисторами, которые действуют как переключатели. Когда ток не протекает, транзистор "выключен" и соответствует биту 0, а когда он протекает, то "включен" и соответствует биту 1.

Проще говоря, биты 0 и 1 как будто соответствуют дыркам, так что пустая дырка - это бит 0, а дырка, занятая электроном, - это бит 1. Поэтому эти устройства называются электроникой. В качестве примера на рисунке 1 показана запись некоторых символов в двоичном виде. Теперь, когда у нас есть представление о том, как работают современные компьютеры, давайте попробуем понять, как работают кванты.

От битов к кубитам

Фундаментальной единицей информации в квантовых вычислениях является квантовый бит или квант. Кубиты по определению являются двухуровневыми квантовыми системами - здесь мы увидим примеры, - которые, как и биты, могут быть на низком уровне, который соответствует состоянию низкого возбуждения или энергии, определяемой как 0, или на высоком уровне, который соответствует состоянию высокого возбуждения или определяемой как 1. Однако, и в этом заключается фундаментальное различие с классическими вычислениями, квадты могут также находиться в любом из бесконечных промежуточных состояний между 0 и 1, например, в состоянии, которое составляет половину 0 и половину 1, или в трех четвертях 0 и четверти 1.



Квантовые алгоритмы, экспоненциально более мощные и эффективные вычисления

Целью квантовых компьютеров является использование этих квантовых свойств *квитов*, как квантовых систем, которыми они являются, для запуска квантовых алгоритмов, которые используют перекрытие и чередование, чтобы обеспечить гораздо большую вычислительную мощность, чем классика. Важно отметить, что реальное изменение парадигмы заключается не в том, чтобы делать то же самое, что делают цифровые или классические компьютеры - нынешние, но более быстрые, как это можно прочитать во многих статьях, а в том, что квантовые алгоритмы позволяют выполнять определенные операции совершенно по-другому, что во многих случаях оказывается более эффективным - то есть, за гораздо меньшее время или с использованием гораздо меньшего количества вычислительных ресурсов.

Давайте посмотрим на конкретный пример. Давайте представим, что мы находимся в Боготе и хотим знать, какой из миллионов вариантов, как добраться до Лимы ($N=1,000,000$), является лучшим. Чтобы использовать компьютеры для поиска оптимального маршрута, необходимо оцифровать 1 000 000 вариантов, что подразумевает их перевод на битовый язык для классического компьютера и в *квиты* для квантового компьютера. В то время как классический компьютер должен будет идти один за другим, анализируя все пути до тех пор, пока не найдет нужный, квантовый компьютер использует процесс, известный как квантовый параллелизм, который позволяет ему рассматривать все пути одновременно. Это означает, что если классическому компьютеру нужен порядок $N/2$ шагов или итераций, т. е. 500 000 попыток, то квантовый компьютер найдет оптимальный путь только после операций с \sqrt{N} в реестре, т. е. 1000 попыток.

В предыдущем случае преимущество квадратично, но в других случаях оно даже экспоненциально, что означает, что при n кубитах мы можем получить вычислительную мощность, эквивалентную 2^n бит. В качестве примера можно привести то, что с помощью примерно 270 кубитов мы могли бы иметь в квантовом компьютере больше базовых состояний - более разных и одновременных символьных строк - чем количество атомов во Вселенной, которое оценивается примерно в 280. Другим примером является то, что с помощью квантового компьютера в диапазоне от 2000 до 2500 кубитов мы могли бы сломать практически всю используемую сегодня криптографию (так называемая криптография с открытым ключом).

Почему важно знать о квантовых технологиях?

Мы находимся в момент цифровой трансформации, когда различные новые технологии, такие как блок-цепь, искусственный интеллект, беспилотники, интернет вещей, виртуальная реальность, 5G, 3D-принтеры, роботы или автономные транспортные средства имеют все больше и больше присутствия в различных областях и секторах. Эти технологии, призванные повысить качество жизни человека, ускоряя развитие и создавая социальное воздействие, сегодня развиваются параллельно. Лишь

в редких случаях мы видим компании, разрабатывающие продукты, использующие комбинации двух или более из этих технологий, такие как блок-цепь и IoT или беспилотные летательные аппараты и искусственный интеллект. Несмотря на то, что им суждено сойтись, тем самым создавая экспоненциально большее влияние, начальная стадия разработки, в которой они находятся, и нехватка разработчиков и людей с техническими профилями означают, что конвергенция все еще остается нерешенной задачей.

Ожидается, что в силу своего разрушительного потенциала квантовые технологии не только сближаются со всеми этими новыми технологиями, но и окажут сквозное воздействие практически на все из них. Квантовые вычисления будут угрожать аутентификации, обмену и безопасному хранению данных, оказывая серьезное влияние на те технологии, в которых криптография играет более значимую роль, такие как кибербезопасность или блок-цепочка, а также оказывают незначительное негативное влияние, но также должны учитываться в таких технологиях, как 5G, IoT или беспилотные летательные аппараты.

Хочешь попрактиковаться в квантовых вычислениях?

Десятки квантовых компьютерных симуляторов уже доступны в сети с различными уже используемыми языками программирования, такими как C, C++, Java, Matlab, Maxima, Python или Octave. Также появились новые языки, такие как Q#, запущенные Microsoft. Вы можете исследовать и играть с виртуальной квантовой машиной через такие платформы, как IBM и Rigetti.

Mini BlocklyChain создан компанией OpenQbit.com, которая специализируется на разработке технологий квантовых вычислений для различных типов частного и государственного секторов.

Почему Mini BlocklyChain отличается от других блок-цепей просто потому, что система была создана по модульному принципу и включает в себя квантовые вычисления.

- (2) <https://blogs.iadb.org/conocimiento-abierto/es/como-funciona-la-computacion-cuantica/>

29. Приложение "Расширенные блоки для базы данных SQLite".

Подробнее о правильном и своевременном использовании каждого блока, входящего в состав расширения OpenQbitSQLite, можно узнать, обратившись к первоначальному автору расширения по следующей ссылке.

OpenQbitSQLite является клоном оригинального дизайна с небольшими модификациями для использования с уровнем безопасности AES.

<https://github.com/frdfsnlght/aix-SQLite>

30.Приложение "Пример создания системы Mini BlocklyChain".

Мы создадим тестовую систему, где сможем проверить функциональность, преимущества и простоту создания системы Mini BlocklyChain.

Мы начнем с проектирования и разработки хранилища, в котором будет храниться информация, которую мы уже определили как цепочку блоков. Проектирование будет основано на базе базы данных SQLite, и в зависимости от каждой организации или проекта она может быть легко изменена другой базой данных, такой как Microsoft SQL Server, MySQL, Maria DB, Oracle, DB2, PostgreSQL и др. Хотя опять-таки благодаря процессу транзакций и согласованию с базой данных SQLite можно использовать на любом уровне и для делового или личного применения.

При создании базы данных, называемой **minibc**, в ней будет располагаться таблица, в которой будет храниться строка блока. Эта база данных будет встроена в конечный программный код, который будет установлен на узлах, это место хранения называется "активы упакованы" - это внутренний вид в блочных средах, таких как App Inventor.

\$ кв.м. минибк.дб

SQLite версия 3.32.2 2020-06-20 15:25:24

Ведите ".help" для подсказок по использованию.

Подключен к базе данных переходных процессов в памяти.

Используйте ".open FILENAME" для повторного открытия в постоянной базе данных.

sqlite> .quit

Проектирование полей таблицы nblock.

КРЕАТКИЙ ТАБЛОК НБЛОК (

id целый первичный ключ АВТОИНКРЕМЕНТА

... "prevhashVARCHAR NOT NULL

НОВЫЙ ХАШВАРЧАРКАРХАРХАРХАРХАРХЕР НЕ НУЛЬШЕ

ntransINTEGER NOT NULL

не-ЦЕНТЕГЕР НЕ НУЛЬШИЙ

... ЦЕЛОЕ ЧИСЛО НЕ НУЛЕВОЕ

);

Мы создали стол **nblock**.

\$ кв.м. минибк.дб

SQLite версия 3.32.2 2020-06-20 15:25:24

Введите ".help" для подсказок по использованию.

Подключен к базе данных переходных процессов в памяти.

Используйте ".open FILENAME" для повторного открытия в постоянной базе данных.

sqlite> .open minibc.db

sqlite> CREATE TABLE nblock (id целый первичный ключ AUTOINCREMENT , предхэш ключ
HE NULL , новый хэш ключ HE NULL , ntrans INTEGER NOT NULL , nonce INTEGER NOT
NULL , qrng INTEGER NOT NULL);

Посмотрим что-нибудь подобное:



```
$ 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
$
```

SQL оператор для
создания таблицы
nblock.

Далее мы создадим исходный хэш системы под названием "Genesis", в основе которого лежит создание нового хэша первого блока цепочки блоков, который мы будем использовать (**NewBlock**). Затем мы создадим второй хэш, который на основе хэша "Genesis" будем называть "единым", потому что он должен соответствовать первому хэшу, потому что тест проверки хэша на цепочке исходных блоков должен всегда соответствовать: prevhash = newhash.

НьюБлок. Для создания хэша "Genesis" будем использовать входные параметры:



Входной параметр: **данные <String>**, **previousHash <String>**.

Выходной параметр: один SHA256 Hash.

В этом случае мы будем использовать в качестве "previousHash" инициалы, равные "0", а в случае с входными данными "данные" - слово "Genensis".

Это даст нам вычисленный хэш комбинации обоих данных:

SHA256 (Genesis0) = eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642

Вышеприведенная строка будет вставлена в nblock таблицу, эта строка должна быть зашифрована, для этого мы используем расширение (**OpenQbitEncDecData**).



Мы будем использовать расширение OpenQbitSSHClient для вставки данных в БД minibc.db, это будет оператор INSERT в БД minibc.db таблицы nblock.

```
sqlite3 keystore.db "вставить в nblock (prevhash, newhash, ntrans, nonce, qrng) значения ('0', 'eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642', '1', '0', '0');";
```

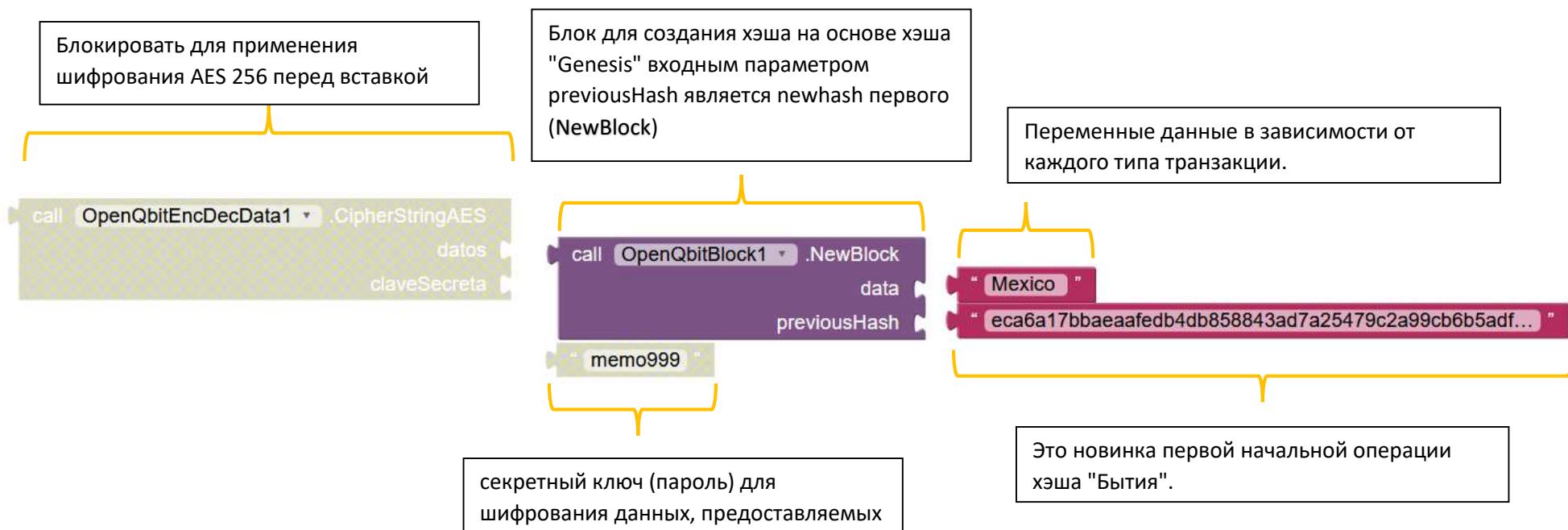
Мы создали "единственный" хэш с оператором SQL на основе хэша "Genensis":

```
sqlite3 клавиатура.db "вставка в nblock (prevhash, newhash, ntrans, nonce, qrng) значений ('eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642', f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78e78ab59a68", "1", "0", "0");"
```

Хэш "один" newhash вычисляется как:

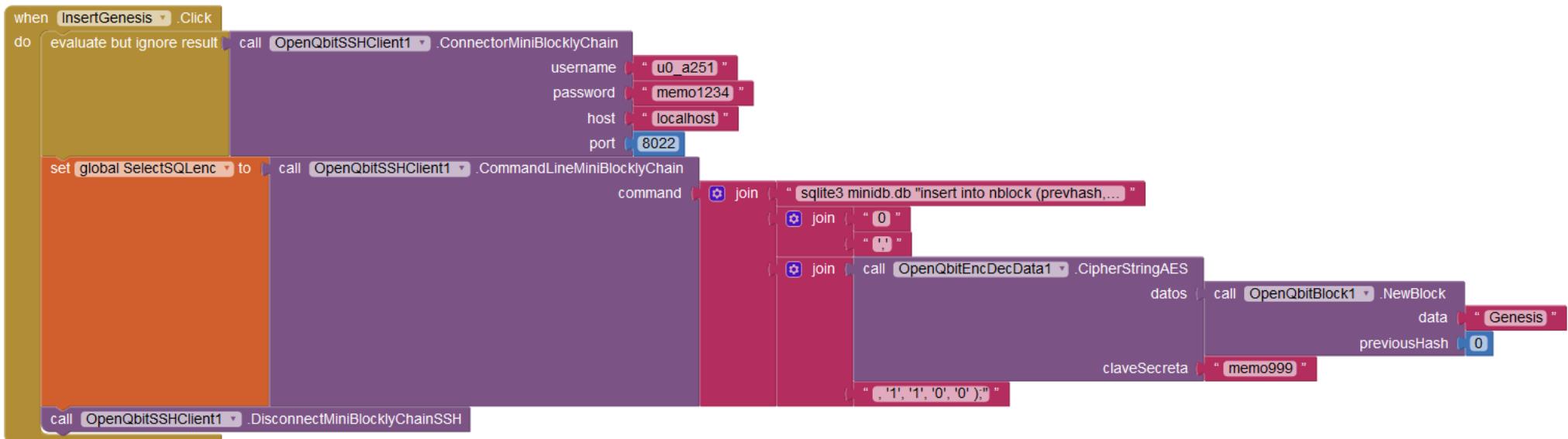
SHA256 (есаба17ббаеаафedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642Mexico) =
f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78e78ab59a68

Второй хэш, основанный на первом инициализационном хэше "Genesis", мы должны сделать два INSERTS в начале, потому что любая начальная проверка блочной цепочки должна соответствовать равенству полей: prevhash (предпоследние данные) = newhash (последние данные).



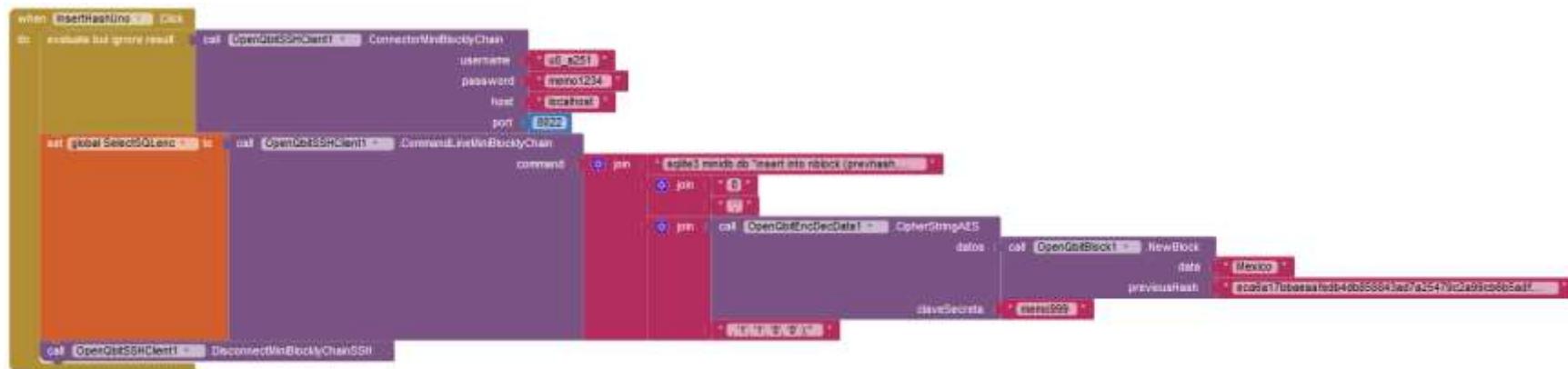
Мы продолжаем с созданием исполнения в App Inventor хэшей выше; хэша "Бытие" и хэша "начало".

Теперь мы покажем, как INSERT хэша "Genesis" будет интегрирован в исходную структуру блочной цепочки внутри таблицы nblock в системе App Inventor:



Так как мы вставили первые данные в nblock-таблицу на основе хэша "Genesis", мы продолжаем делать ссылку на второй INSERT на "один" хэш.

Он интегрирует INSERT хэша "One" в исходную структуру блочной цепочки внутри таблицы nblock в системе App Inventor:



Две предыдущие структуры программирования (хэш "Genesis" и хэш "one") должны быть интегрированы начальным узлом системы Mini BlocklyChain, важным моментом является то, что база данных minibc.db со всей ее внутренней структурой (таблицами) будет реплицироваться через сеть Mini BlocklyChain, основанную на архитектуре "Peer to Peer".

К настоящему времени мы завершили создание базовой и фундаментальной структуры хранилища блоков, и оно готово принимать новые блоки из будущих транзакций, возникающих или запрашиваемых в системе.

Мы уже вставили первые хэш-данные "Genesis" и хэш-"one" в нашу базу данных **minibc.db**, теперь мы будем использовать следующие SQL-запросы для запроса полей **prevhash** и **newhash** в таблице **nblock**.

Этот момент будет иметь фундаментальное значение, так как на основе сравнения этих двух полей мы узнаем, является ли блок-цепочка последовательной и поддерживает ли она целостность и безопасность операций. Это всего лишь стартовый механизм для просмотра цепочки блоков, так как в будущем мы рассмотрим использование блока для вычисления дерева мерклей, что станет еще одним важным инструментом для обеспечения целостности и безопасности цепочки блоков и в нашей системе.

Пока мы будем рассматривать только структуру или SQL предложения, которые мы должны использовать, как мы это делали ранее с расширением ([OpenQbitSSHClient](#)), с ними мы сможем ознакомиться с полями **prevhash** и **newhash**. Позже мы сможем делать простое сравнение равных данных для этой проверки каждый раз, когда мы будем добавлять новый блок.

Для предсказания:

```
sqlite3 minibc.db "SELECT prevhash FROM nblock ORDER BY ID DESC LIMIT 1";".
```

Для Ньюхэша:

```
sqlite3 minibc.db "ВЫБЕРИТЬ НОВЫЙ ХАШ ОТ nblock ЗАКАЗЧИКА ПО ИДЕЙСТВУЮЩЕМУ DESC LIMIT 1;"
```

Теперь мы сконцентрируемся на том, как мы будем запрашивать новую сделку и/или транзакцию, которая будет вставлена в очередь транзакций.

Существуют два шага в качестве требований к отправке транзакции в очередь транзакций.

Первое требование заключается в том, чтобы на балансе нашего счета было достаточно "активов", чтобы покрыть сумму, подлежащую отправке. Помните, что "активы" могут быть не только числом или суммой, но мы можем создавать "активы" любого вида в зависимости от каждой создаваемой системы.

Примеры активов:

- ✓ Внутреннее количество "виртуального или крипто-валютного" количества нового происхождения.
- ✓ Данные, относящиеся к цифровым данным (все типы документов)
- ✓ Хэширует подписи аутентификации для строк или всех типов файлов.
- ✓ Любые операции или передача цифровой информации или ее представление.

Баланс "активов" каждого узла получается с помощью блока (**GetBalance**).



Второе требование заключается в том, чтобы при отправке транзакции были указаны минимальные параметры, а именно

- ✓ Адрес источника. - это адрес, с которого будут отгружены активы.
- ✓ Адрес назначения. - это адрес пользователя, который получит отправленные средства.
- ✓ Активный. - Это данные с собственным значением, заданным в каждой системе, которые будут отправляться в каждой транзакции.

Вышеуказанные адреса (origin-destination) соответствуют публичным ключам каждого пользователя, когда были созданы учетные записи каждого пользователя. Помните, что при создании учетной записи пользователя создаются два типа открытых и закрытых ключей.

Публичный ключ - это адрес, который будет совместно использоваться всей системой, и который любой пользователь системы может видеть и использовать для отправки или получения транзакций.

Частный ключ - это адрес, который используется локально каждым узлом, и, как показывает его имя, он предназначен для частного использования, что помогает создавать цифровые подписи для обеспечения безопасности отправляемых транзакций, этот ключ никогда не должен использоваться совместно, и он предназначен для локального использования и уникален для узла-источника.

Для использования предыдущих параметров мы будем опираться на два репозитория, где хранятся адреса "закрытых ключей", которые находятся в базе данных keyystore.db и будут локально использоваться каждым узлом без совместного использования в системе, а другой репозиторий, где хранятся все адреса "открытых ключей", которые находятся в базе данных publickeys.db, будет совместно использоваться по протоколу "Peer to Peer" во всех узлах системы.

БД "keyystore.db" и "publickeys.db" уже созданы в приложении "Создание баз данных KeyStore & PublicKeys".

В разделе "Установка и настройка сети RESTful Mini Sentinel" уже создана база данных и система для очереди транзакций. Там, где мы уже создали базу данных для транзакций, называемую op.sqlite3 , у нас есть две таблицы, одна из которых является "транзакцией", которая заботится о транзакциях (**очередь транзакций**), а другая называется "подписью", где хранятся цифровые подписи каждой операции.

Система SQLite RESTful является сетью резервного копирования в этом случае мы будем использовать ее для простоты и для тестирования системы резервного копирования, однако, чтобы изменить и использовать ту же базу данных op.sqlite3 в модели "Peer to Peer", нам придется использовать базу данных в общей модели снова только в системе синхронизации, это мы увидим позже.

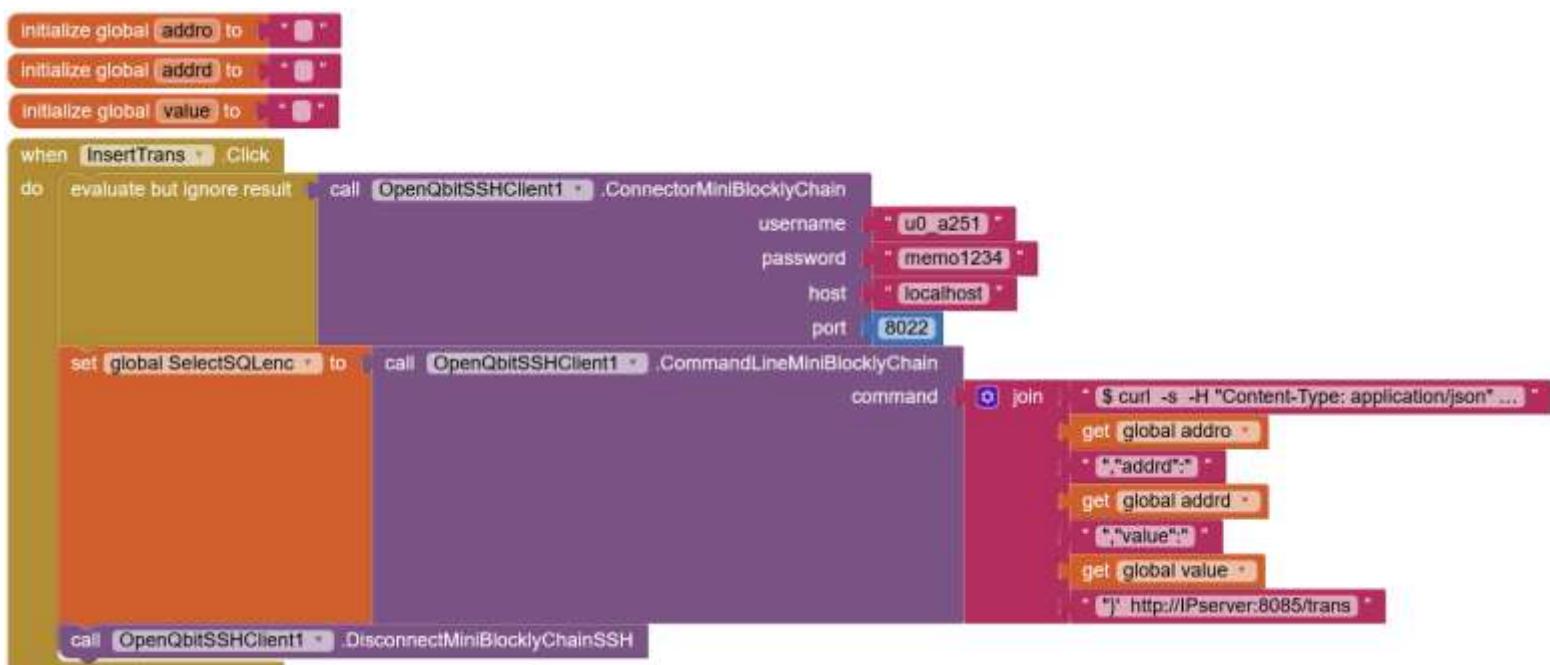
Мы начнем посыпать операции в очередь транзакций, используя RESTful, применяемый к базе данных op.sqlite3

Мы создали новую транзакцию в таблице "Транзакции".

```
$ curl -s -H "Content-Type: application/json" -d '{"addr":  
"MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9", "addrd":  
"MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh", "value": "999"}' http://IPserver:8085/transactions
```

```
# выходы  
{  
    "id": 1,  
    "Адрес": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",  
    "адрд": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh",  
    "стоимость": "999"  
}
```

Внедрение в App Inventor:



Входные параметры: addro, addrd, value - переменные, которые можно ввести в алгоритм и извлечь из базы данных keystore.db.

См. приложение "Создание базы данных KeyStore".

Теперь мы вставляем цифровые подписи из предыдущей сделки. В таблице "Знак"

```
$ curl -s -H "Content-Type: application/json" -d '{  
  "trans_id":1  
  "знак": "59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62",  
  "хэш": "f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68"}'  
http://IPserver:8085/sign  
  
# выходы  
{  
  "id": 1,  
  "trans_id": 1,  
  "знак": "59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62",  
  "хэш": "f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68"  
}
```

ПРИМЕЧАНИЕ: Цифровая подпись (подпись) должна быть получена до отправки команды скручивания, она генерируется блоком (**GenerateSignature**).

Входные параметры: Trans_id, знак, хэш - это переменные, которые могут быть введены в алгоритм, а с помощью блока (**GenerateSignature**) будет сгенерирована цифровая подпись транзакции.

Теперь мы рассмотрим процедуру генерации цифровой подписи, которая будет применяться к каждой совершенной сделке.

Генерация цифровой подписи для сделки. Когда мы используем блок (**GenerateSignature**), у нас будет в результате тип файла .sig этот файл, который мы будем использовать для сохранения в поле знака в таблице "знак", эта сигнатура будет использоваться при обработке очереди транзакций, и это еще один параметр для обеспечения безопасности между отправителем и получателем, а также количество или тип транзакций между ними.

Для работы с бинарными данными подобно файлу file.sig нам придется преобразовать их в base64, а затем сохранить в переменной знака таблицы "Знак". Для этого мы будем использовать блок (**EncoderFileBase64**).

Как рассчитывается значение поля хэша для таблицы "Знак" каждой транзакции:

Транзакция Hash = SHA256(Адрес источника + Адрес назначения + Актив + fileBase64.sig)

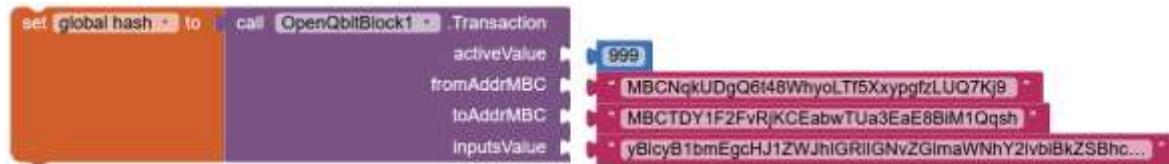
Пример хэша типа SHA256:

SHA256(MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9 +

MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh + 999 +).

Выход: 9d45198faaef624f2e7d1897dd9b3cedebесса7fbac516ed1756b350fe1d56b4

Для вычисления хэша нам придется опираться на Блок(**Транзакцию**).



Выходным параметром является хэш, который будет храниться для каждой транзакции, посыпаемой с любого узла в системе. Он хранится в поле знака таблицы "Знак" в базовой op.sqlite

В предыдущей операции мы убедились, что только уникальный и неповторяющийся хэш будет представлять собой каждую транзакцию, отправленную в очередь, и этот представительный хэш является совокупностью переданной информации.

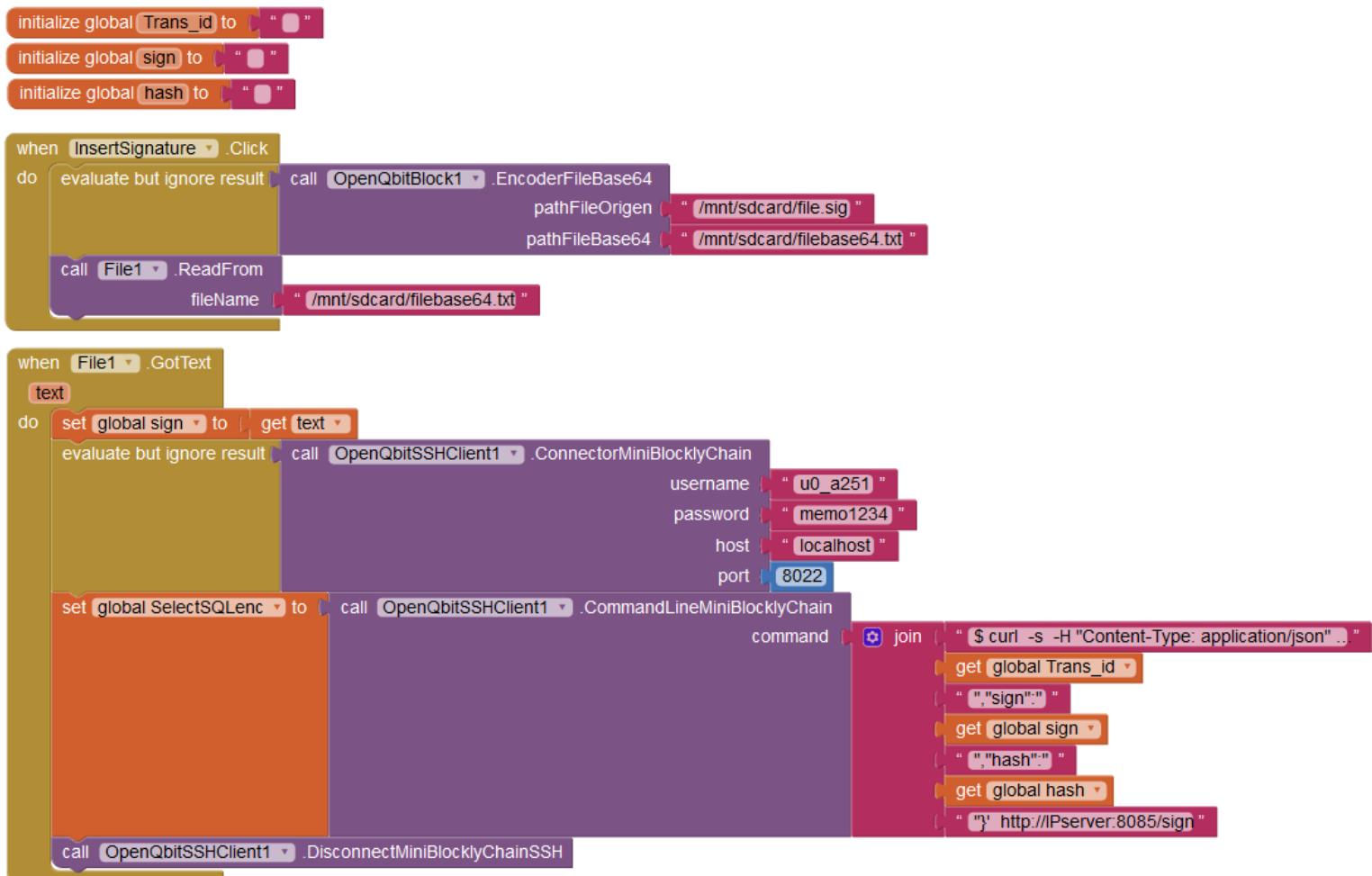
Единственное, чего не хватает - это включения переменной Trans_id, которая может быть идентификатором для различия, какой узел посыпается для каждой транзакции. В нашем примере мы поместим переменную Trans_id с фиксированным значением "1". Потому что это первый узел, который настраивается в нашей сети. Это значение может варьировать синтаксис в соответствии с каждой конкретной конструкцией, поэтому для простоты мы выбрали простой идентификатор.

Так как мы определили все переменные, мы создадим структуру и последовательность выполнения блока в App Inventor, чтобы выполнить INSERT в таблице "Знак".

ПРИМЕЧАНИЕ: Важно учитывать, что каждая отправка транзакции состоит из двух INSERTS в БД op.sqlite3, одна из которых должна быть выполнена в таблице "trans", а соответствующие значения - в таблице "sign".

При проектировании базы данных op.sqlite3 были созданы две отдельные таблицы в связи с тем, что в будущем можно будет зашифровать только таблицу "Знак", так как она содержит информацию, которая не должна передаваться в сети плоским способом, этот вариант оставляется на рассмотрение каждой будущей конструкции.

Создадим структуру исполнения блоков и методов внутри приложения Inventor INSERT в таблице "Знак".



К этому моменту мы уже закончили INSERT к таблице "Знак" и к таблице "Транс". Они находятся внутри БД `op.sqlite3`, и данные, вставленные в эти две таблицы, будут использованы для создания очереди транзакций, которая будет отправлена всем узлам для ее обработки.

На данный момент мы будем использовать Java SQLite-Redis коннектор, который будет выполнять преобразование данных из базы данных `op.sqlite3` в базу данных Redis. См. приложение "Java SQLite-Redis Code Connector".

После реализации SQLite-Redis Connector очередь транзакций будет доставлена во все узлы системы через систему Redis.

Мы начнем процесс того, как мы можем получить очередь транзакций должным образом, чтобы иметь возможность ее обработать.

Очередь транзакций будет доставляться через службу Redis. Это база данных в реальном времени, которая имеет соответствующие блоки управления в среде App Inventor.

Конфигурация среды Redis в блочной системе App Inventor.

Важным моментом является то, что блоки управления сервером Redis до момента написания данного руководства доступны только в системе App Inventor. В случае использования системы Blockly, отличной от App Inventor, вам придется использовать Redis CLI (Command-Line), выполняя его через отправку команд на терминал Termux через расширение ([OpenQbitSSHClient](#)).

Пример использования в Redis CLI (командная строка):

Получить все **этикетки** или ключи на Redis.

\$ redis-cli KEYS *

Чтобы получить ценность от ключа.

\$ redis-cli GET <key>

В нашем случае, поскольку мы используем App Inventor, мы рассмотрим, как использовать блоки для работы с Redis.

Внутри App Inventor у нас есть несколько блоков для использования с Redis, мы начали создавать новый проект мы будем: Мои проекты > Начать новый проект



После создания проекта переходим в левый верхний колонтитул и в разделе "Палитра" нажимаем "Хранилище" и перетаскиваем объект "CloudDB", в который интегрируются все возможности для настройки подключения к серверу Redis.



Конфигурация очень проста, нам нужно лишь задать следующие параметры, чтобы иметь возможность подключиться к нашему серверу Redis (Local), который запущен в нашем узле (Мобильный телефон).

ProjectID. - Это ID, с которого будет начинаться метка, идентифицирующая очередь транзакций в Redis.

RedisPort. - Это порт сервера Redis, к которому мы будем подключаться по умолчанию 6379.

RedisServer. - В нашем случае это домен или локальный IP узла, и он будет 127.0.0.1.

Токен. - Это пароль сервера Redis, разрешенного для подключения.

UseSSL. - Эта опция дает нам возможность использовать SSL-сертификаты в нашем случае, если мы оставляем ее недоступной.

В нашей модели и проектировании системы мы будем иметь следующие параметры во всех узлах сети.



Пришло время просмотреть блоки, которые обеспечивают нам контроль и обработку данных в среде базы данных Redis.

Начнем с блока методов (**DataChanged**).



Этот метод будет давать нам два значения каждый раз, когда происходит изменение в сервере Redis, который мы настроили:

Тэг. - Это значение является тегом, идентифицирующим очередь транзакций.

Значение. - Данное значение содержит список всех транзакций, отправленных на обработку. Это значение представляет собой строковый список типа `<String>`.

В случае с "значением" мы начнем обрабатывать информацию следующим образом.

По мере того, как вы предоставляете нам цепочку всех операций с хэш-значениями, мы начинаем с проверки того, является ли эта цепочка действительной, и проверяем, не была ли она изменена с момента ее возникновения. Для этого мы используем очень полезный алгоритм проверки больших объемов информации.

Мы используем алгоритм дерева Меркла. Применение этого алгоритма дает нам уверенность в целостности получаемой информации (очередь транзакций).

Давайте посмотрим следующий пример очереди транзакций в Redis, мы выполняем Redis CLI (Command-Line) из терминала Termux для проверки ключа "LATAM:HASH", мы должны были уже выполнить SQLite-Redis коннектор, чтобы иметь возможность обратиться к этому ключу.

```
C:памятка>редис-кли
127.0.0.1:6379> получить LATAM:HASH
"9d45198faaef624f2e7d1897dd9b3cde6ecca7fbac516ed1756b350fe1d56b4,"
f71c801a5fd25fc303ebc8c616204b4877ffb93006ec6a88bc30acf43ec250f5,"\60f8a3bcac1
ea7d38e86efbc3e3e00480807d23f980391000766e804ed14ecb2 ,
8a6dfe1d38c22e0f9212052efa6136da3edf1fb1b2a3e25224ac3d689124b754"]
127.0.0.1:6379>
```

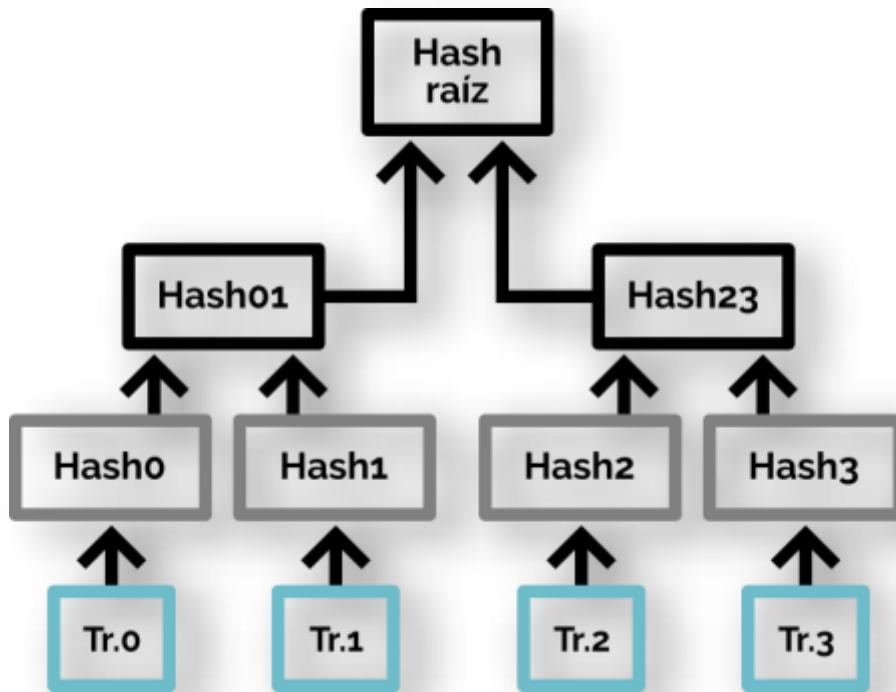
Предыдущая очередь транзакций имеет только три элемента, так как мы видим четыре хэша, представляющие отдельную транзакцию, из которых она состоит, и они являются хэшами каждой транзакции, которая была изначально введена в БД op.sqlite3 в таблицах "trans" и "sign".

Теперь пришло время понять, как работает дерево меркл.

Дерево хэш-меркл - это бинарная или небинарная структура дерева данных, в которой каждый узел, не являющийся листом, помечен хэшем конкатенации меток или значений его дочерних узлов. Они являются обобщением хэш-списков и хэш-строк.

В нашем случае для вычисления дерева меркл для четырех элементов мы сделаем следующий вычисление: вычислим результат взятия пар данных, сцепленных между собой, и получим их соответствующие хэши, результаты первого уровня будут применены к результатам второго уровня до тех пор, пока не будет найден только один конечный элемент, который будет называться хэш мерклерут (корневой хэш).

Рассмотрим следующую диаграмму, которая описывает этот процесс.



Очередь транзакций, основанная на хэш-тестах, будет вытаскиваться через блок (`GetMerkleRoot`).



Корень хэша:
51431822de7c94b90dc06d47b8f6275f315a4976c8479d30c32747fa90325432

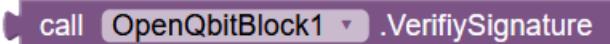
Затем результат сравнивается с ключом LATAM:merkleroot в локальной системе Redis и оба ключа должны совпадать для проверки целостности данных.

Другая точка безопасности, предоставляемая деревом меркл, заключается в подтверждении того, что конкретная транзакция включена в очередь транзакций с момента ее возникновения, и что она не была введена каким-либо внешним или внутренним средством связи мошенническим путем.

В случае, если цепочка состоит из нечетных элементов в ее суммировании, то последний элемент дублируется, чтобы иметь расположение и начать выполнение алгоритма.

Еще одним не менее важным моментом является время для подтверждения того, что каждая операция соответствует месту ее отправления и что активом является тот, который был отправлен по адресу отправления.

Здесь находится Блок (VerifySignature).



Перед выполнением предыдущего блока необходимо сначала загрузить файл (file.sig) в двоичном формате из таблицы "Знак":

```
sqlite3 op.sqlite3 "выбрать знак из знака where=id_addr;"
```

id_addr: Это идентификатор исходного адреса таблицы "trans".

Предыдущий поисковый запрос выдаст нам данные в формате Base64 цифровой подписи текущей транзакции, для преобразования их в оригинальный формат (двоичный) нам понадобится Block(**DecoderFileBase64**).

Поскольку у нас есть наш двоичный файл (file.sig), мы должны будем загрузить в систему открытый ключ источника и открытый ключ получателя также в двоичном формате, наличие четырех данных в их соответствующих форматах станет временем для запуска проверки цифровой подписи в системе.

- ✓ Домашний адрес с открытым ключом
- ✓ Адрес открытого ключа получателя
- ✓ Активная отправка.
- ✓ Цифровая подпись (file.sig)

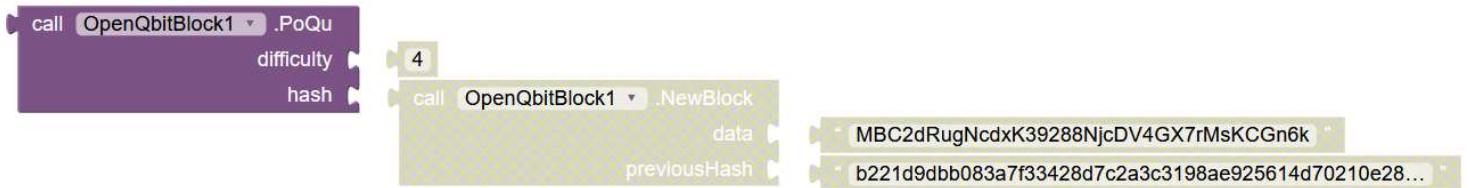
Открытые ключи в двоичном формате могут быть загружены в соответствующем формате (двоичном) из общей базы данных publickeys.db.

Этот процесс должен выполняться для каждой отдельной операции в очереди транзакций.

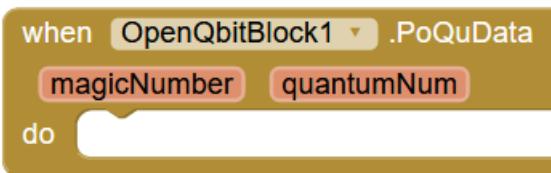
Наконец, мы рассмотрим, как система выбирает узел по взаимному согласию, чтобы иметь возможность добавить следующий блок в цепочку блоков и быть тем, кто обрабатывает очередь транзакций.

Такой способ выбора выигрышного узла для обработки очереди транзакций основан на нашем алгоритме, разработанном для системы Mini BlocklyChain.

Как мы реализовали консенсус PoQu "Доказательство Кванта". Этот процесс консенсуса основан на генерации квантовых случайных чисел и применяется с использованием блока (PoQu).



Сначала мы получаем два параметра, которые блок (PoQu) дает нам в методе (**PoQuData**), параметры - **magicNumber** и **quantumNum**.



Магический номер - это число "nonce", это целое число, которое получается, делая внутренний PoW с трудом не больше 5, это число отвечает за то, чтобы дать первое требование к узлу с магическим номером, узел сможет выполнить требование о получении числа системы генерации квантовых случайных чисел, которое находится в диапазоне от 0 до 1, это число даст вероятность, установленную случайным образом для узла во время выполнения, которая будет храниться в таблице под названием "голосование".

В таблице "голосование" будет храниться **квантоваяNum**, рассчитанная по каждому узлу, это хранение будет производиться с несколькими узлами до тех пор, пока не будет установлено определенное время в каждом проекте системы, рекомендуется иметь записи $((N/2) + 1)$, где "N" - это количество узлов, доступных в системе, и оно может быть установлено или контролироваться действием в "cron" инструменте управления задачами каждого узла.

Важным моментом является то, что к этому моменту уже должна быть установлена локальная синхронизация времени каждого узла через автоматическую систему

мобильного телефона в случае использования сети "**Peer to Peer**" при передаче очереди транзакций.

Конфигурация агента Крона в узлах. См. раздел "Синхронизация времени в узлах системы (Мобильный телефон) в минутах и секундах".

В данном примере, поскольку мы используем резервную сеть связи, синхронизация минут и секунд для узлов не требуется, так как наш пример занимает схему "**Клиент-сервер**", этот вид связи находится только в процессе передачи очереди транзакций. Все остальные процессы между узлами проходят через коммуникацию "**Peer to Peer**".

Теперь мы рассмотрим структуру, дизайн и создание таблицы "голосование", которая будет расположена в базе данных под названием "quorum.db", которую мы также будем создавать в этом примере.

\$ kv.3

SQLite версия 3.32.2 2020-06-20 15:25:24

Введите ".help" для подсказок по использованию.

Подключен к базе данных переходных процессов в памяти.

Используйте ".open FILENAME" для повторного открытия в постоянной базе данных.

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

Создание такой же таблицы **голосования** показано ниже, но это делается путем введения SQL-оператора в сегментированном виде:

\$ kv.3

SQLite версия 3.32.2 2020-06-20 15:25:24

Введите ".help" для подсказок по использованию.

Подключен к базе данных переходных процессов в памяти.

Используйте ".open FILENAME" для повторного открытия в постоянной базе данных.

sqlite> .open quorum.db

sqlite> CREATE TABLE vote (

...> id целочисленный первичный ключ AUTOINCREMENT

...> node_imei VARCHAR NOT NULL,

...> quantumNum INTEGER NOT NULL,

...> ...nonce INTEGER NOT NULL

...>);

sqlite> .таблицы

голосовать

```
sqlite> .quit
```

Мы увидим нечто подобное при создании базового "quorum.db" и табличного голосования.



The screenshot shows a terminal window on an Android device. The title bar indicates the battery level is at 26% and the time is 9:02 p.m. The terminal output is as follows:

```
$ 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
$
```

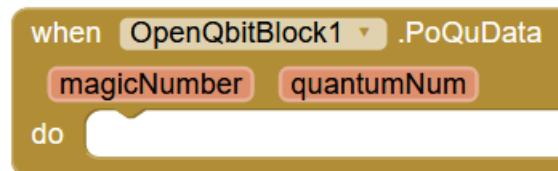
Below the terminal window is a standard Android keyboard.

Теперь посмотрим, как мы получим значения из таблицы "голосование".

node_imei. - Мы **получаем** это значение с помощью блока (**GetDeviceID**).



quantumNum - Это значение является одним из результатов метода (**PoQuData**), который получается с помощью блока (**PoQu**).



нонс. - Это значение получается из того, что блок (**PoQu**) уже выполнен в интегральном виде и совпадает с магическим номером метода (**PoQuData**).

После заполнения INSERTS в таблице "голос" необходимо сделать копию БД.

По прошествии определенного времени и на основе дизайна каждой системы, на каждом сетевом узле будет выполнен сервис "cron", а следующий SELECT будет обработан в таблице "голос":

```
SELECT node_imei FROM vote ГДЕ волшебный номер= (SELECT
max(magicNumber) FROM vote);
```

Предыдущий SELECT возвращал IMEI результат с большей вероятностью, теперь каждый узел, на котором запущен SELECT, будет сравнивать свой IMEI с IMEI результатом, и только тот узел, который совпадает, создаст файл с наибольшей вероятностью, который будет реплицирован в сети "Peer to Peer" с помощью файла в формате IMEI.mbc, который будет содержать IMEI узла-победителя.

Победивший узел сможет начать обработку очереди транзакций. Основываясь на всех вышеперечисленных блоках.

Два важных момента состоят в том, что в зависимости от создания каждой системы три процесса должны быть пересмотрены и настроены каждым проектировщиком.

1.- Когда выигрышный узел начинает обрабатывать очередь транзакций, должен быть реализован метод или процесс, при котором необходимо проверить, что выигрышный узел находится в режиме онлайн и связь с сетью не была потеряна.

2.- Когда начинается обработка очереди транзакций, выигравший узел должен запустить два управляющих флага, указывающих на начало обработки и еще один, подтверждающий, что обработка очереди транзакций завершена. Эти два флага

должны быть разделены в сети между всеми узлами, это поможет обнаружить сбои в соединении или обработке, или слишком много времени обработки.

3.- В случае сбоя связи или другого события, когда выигравший узел не смог обработать очередь транзакций, следующий узел должен быть выбран в диапазоне непосредственной вероятности.

Предыдущим пунктом можно управлять с помощью службы, которая проверяет, что выигравший узел подключен к сети и может использовать службу "cron". Сценарий должен быть разработан для каждого разработанного случая. Однако ниже показан общий пример сценария оболочки, чтобы иметь возможность быть изменены в соответствии с потребностями каждой системы Mini Blocklychain.

```
#!/bin/bash
dir="/data/data/com.termux/files/home/Sync/imei";
if [ !"$(ls $directorio)" ]
then
    sqlite3 quorum.db "UPDATE vote SET magicNumber=0 WHERE magicNumber= (SELECT
max(magicNumber) FROM vote);"
else
    MAX_NUM=$(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) // Usar el bloque (GetDevice)
    if [ IMEI_quorum -eq IMEI_local ]
    then
        touch $MAX_NUM > IMEI.mbc
    fi
fi
exit
```

31. Приложение "Интеграция со средами Ethereum и Биткойн".

Теперь мы увидим, как мы можем интегрировать две наиболее известные в мире системы блокчейнов, специализирующиеся на криптовалютах, такие как Ethereum и Bitcoin.

Начнем с установки программного обеспечения, которое поможет нам проводить все возможные транзакции в среде Ethereum.

Что такое Ethereum?

Ethereum - платформа с открытым исходным кодом, децентрализованная в отличие от других блокчейнов, Ethereum может сделать гораздо больше. Он программируемый, что означает, что разработчики могут использовать его для создания новых типов приложений.

Эти децентрализованные приложения (или «dapps») пользуются преимуществами криптовалюты и технологии блокчейна. Они надежны и предсказуемы, что означает, что, как только они будут «загружены» в Ethereum, они всегда будут работать в соответствии с графиком. Они могут контролировать цифровые активы для создания новых типов финансовых приложений. Они могут быть децентрализованы, что означает, что ни одно юридическое или физическое лицо не контролирует их.

Прямо сейчас тысячи разработчиков по всему миру создают приложения на Ethereum и изобретают новые типы приложений, многие из которых вы можете использовать сегодня:

- Криптовалютные кошельки, которые позволяют совершать дешевые и мгновенные платежи с помощью ETH или других активов.
- Финансовые приложения, которые позволяют заимствовать, одолживать или инвестировать ваши цифровые активы
- Децентрализованные рынки, которые позволяют вам обмениваться цифровыми активами или даже обмениваться «прогнозами» о событиях в реальном мире.
- Игры, в которых у вас есть игровые активы и вы даже можете выиграть реальные деньги.
- У него есть умные контракты, которые представляют собой программы с соглашениями, которые должны выполняться при выполнении условий, в которых оно было создано или создано.

Умные контракты имеют сходство с DApps (децентрализованными приложениями), но они также разделены некоторыми важными отличиями.

Как и смарт-контракты, DApp - это интерфейс, который соединяет пользователя со службой провайдера через децентрализованную одноранговую сеть. Но, в то время как умные контракты требуют создания фиксированного числа участников, DApps не имеют пользовательских ограничений. Кроме того, они не просто сводятся к финансовым приложениям, таким как умные контракты - DApp может иметь любую цель, о которой можно подумать.

В нашем случае мы будем использовать библиотеку под названием «Web3j», которая разработана на Java и которая позволяет нам взаимодействовать с блокчейном Ethereum простым и интуитивно понятным способом.

Мы выполняем следующую команду для установки "Web3j":

```
$ curl -L get.web3j.io | sh
```

Затем мы должны создать переменную окружения `$ JAVA_HOME` с путем, где находится исполняемый файл "java", поскольку библиотека будет искать эту переменную для успешного выполнения.

```
$ JAVA_HOME= /data/data/com.termux/files/usr/bin
```

После того, как переменная была создана, нам нужно будет перейти в каталог, в который была установлена библиотека «Web3j», выполнив следующую команду, важным моментом является то, что каталог скрыт после команды «cd», которую мы поставили «». А затем каталог без пробелов, как показано ниже:

```
$ cd .web3j
```

Оказавшись внутри, мы сразу же проверяем, правильно ли работает библиотека, с помощью следующей команды:

```
$ ./web3j versión
```

Это дает нам нечто очень похожее на:

```
$ ls
source.sh web3j web3j-4.5.16
$ ./web3j version

Version: 4.5.16
Build timestamp: 2020-03-06 14:13:49.943 UTC
$
```

Позже мы создадим кошелек, чтобы использовать его в блокчейн-среде Ethereum следующим образом:

```
$ ./web3j wallet create
```

Предыдущая команда дает нам адрес эфириума:

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
$
```

Он дает нам файл формата JSON, в котором он содержит адрес и зашифрованные данные, чтобы впоследствии их можно было использовать для создания закрытого ключа только что созданной учетной записи.

Этот файл JSON будет создан в каталоге по умолчанию, который называется keystore.

Отныне мы уже можем выполнять операции, используя и / или выполняя команду Web3j.

Выше может быть сделано с помощью расширения (ConnectorSSHClient).

Чтобы узнать, как использовать различные параметры и операции библиотеки "Web3j", мы можем помочь себе, ознакомившись с документацией на ее официальном сайте.

<https://docs.web3j.io/>

Для среды Биткойн у нас есть две опции, использующие block (), который генерирует учетную запись Биткойн и соответствующие ей открытый и закрытый ключи, мы можем использовать это для интеграции в среду Биткойн следующим образом, как мы устанавливаем библиотеку Java «Биткойн».

\$ npm install bitcoinj

```
$ npm install bitcoinj
+ bitcoinj@0.0.0
added 1 package from 1 contributor and audited 2
09 packages in 20.528s
$
```

Для использования этой библиотеки мы будем полагаться на ее официальный сайт.

<https://bitcoinj.github.io/>

Второй вариант, позволяющий интегрироваться в среду цепочки блоков Bitcoin, мы можем реализовать, установив пакет Bitcoind для Termux, как показано ниже.

\$ 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-pac
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 directo
ries currently installed.)
Preparing to unpack .../bitcoin_0.20.0_arm.deb .
.
Unpacking bitcoin (0.20.0) ...
Setting up bitcoin (0.20.0) ...
$
```

Теперь, когда мы запустим агент «bitcoind» на терминале Termux, в каталоге будет автоматически создан адрес:

/data/data/com.termux/files/hme/.bitcoin

В этом случае с биткойнами мы будем поддерживать следующую документацию от агента «биткойн».

В этом случае мы также можем использовать расширение (ConnectorSSHCliente) для запуска агента «bitcoind» в зависимости от каждой потребности.

Описание параметров для использования с биткойнами.

НАЗВАНИЕ

bitcoind - запускает Биткойн-Core Daemon

СИНТАКСИС

bitcoind [опции] Запустить Bitcoin Core Daemon

ОПИСАНИЕ

Запустить Bitcoin Core Daemon

ПАРАМЕТРЫ

-?

Распечатать это справочное сообщение и выйти

-alertnotify = <cmd>

Запустите команду, когда будет получено соответствующее предупреждение или мы увидим очень длинный ответвление (% s в cmd заменяется сообщением)

-assumevalid = <hex>

Если этот блок находится в цепочке, предположите, что он и его предки действительны и потенциально могут записать подтверждение соты (0, чтобы проверить все значения по умолчанию:

00000000000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee, testnet:
000000000000000037a8cd3e06cdbb6cc5b5)

-blocknotify = <cmd>

Запустите команду при изменении лучшего блока (% s в cmd заменяется хэшем блока)

-blockreconstructionextratxn = <n>

Дополнительные транзакции для хранения в памяти для компактных блоков (по умолчанию: 100)

-blocksdir = <dir>

Укажите каталог блоков (по умолчанию: <datadir> / blocks)

-только в блоке

Отклонять ли транзакции от пиров в сети. Портфолио или транзакции RPC не затрагиваются. (по умолчанию: 0)

-conf = <файл>

Укажите файл конфигурации. Относительные пути будут обозначены префиксом местоположения датадира. (по умолчанию: bitcoin.conf)

-daemon

Беги в фоновом режиме как демон и принимай команды

-datadir = <dir>

Укажите каталог данных

-dbcache = <n>

Максимальный размер кэша базы данных <n> MiB (от 4 до 16384, по умолчанию: 450). Также для этого кэша используется неиспользуемая память mempool (см. -Maxmempool).

-debuglogfile = <файл>

Укажите местоположение файла журнала отладки. Относительные пути будут иметь префикс с определенным местоположением сетевого каталога данных. (-nodebuglogfile для отключения; по умолчанию: debug.log)

-includeconf = <файл>

Укажите дополнительный файл конфигурации относительно пути -datadir (можно использовать только из файла конфигурации, но не из командной строки)

-loadblock = <файл>

Импортировать блоки из внешнего файла blk000 ???. Dat при запуске-maxmempool=<n>

Храните пул памяти транзакций ниже <n> мегабайт (по умолчанию: 300)

-maxorphantx = <n>

Хранить в памяти не более <n> отключаемых транзакций (по умолчанию: 100)

-mempoolexpiry = <n>

Не храните транзакции в mempool более <n> часов (по умолчанию: 336)

-par = <n>

Установите количество потоков проверки скриптов (от -6 до 16, 0 = авто, <0 = освободить все ядра, по умолчанию: 0)

-persistmempool

Если сохранить Mempool при выключении и загрузить его при перезагрузке (по умолчанию: 1)

-pid = <файл>

Укажите файл PID. Относительные пути будут иметь префикс с определенным местоположением сетевого каталога данных. (по умолчанию: bitcoind.pid)

-punch = <n>

Уменьшите потребности хранения, разрешая сокращение (удаление) старых блоков. Это позволяет вызывать цепочку сокращения RPC для удаления определенных блоков и позволяет автоматически сокращать старые блоки, если в MIB указан целевой размер. Этот режим несовместим с -txindex и -rescan. Предупреждение: для возврата этого параметра требуется повторная загрузка всей цепочки блоков. (по умолчанию: 0 = отключить удаление блоков, 1 = разрешить удаление вручную через RPC,> = 550 = автоматически удалять файлы блоков, чтобы они оставались ниже целевого размера, указанного в MIB)

-reindex

Восстанавливает состояние строки и блочный индекс файлов blk *.dat на диске.

-reindex-chainstate

Восстановите состояние цепочки из текущих проиндексированных блоков. В режиме сокращения или если блоки на диске могут быть повреждены, используйте вместо этого полное переиндексирование.

-sighs

Создавать новые файлы с системными разрешениями по умолчанию вместо umask 077 (действует только при отключенной функциональности кошелька)

-txindex

Поддерживать полный индекс транзакции, используемый вызовом gRPC getrawtransaction (по умолчанию: 0)

-версия

Версия для печати и выхода

Варианты подключения:

-addnode = <ip>

Добавьте узел для подключения и попытайтесь сохранить соединение открытым (см. Справку RPC-команды "addendum" для получения дополнительной информации). Эта опция может быть указана несколько раз, чтобы добавить несколько узлов.

-banscore = <n>

Порог для отключения неправильно работающих пиров (по умолчанию: 100)

-В это время...

Количество секунд, чтобы предотвратить повторное соединение неправильно работающих пиров (по умолчанию: 86400)

-bind = <addr>

Придерживайтесь указанного адреса и всегда слушайте его. Используйте [host]: обозначение порта для IPv6

-connection = <ip>

Подключаться только к указанному узлу; -noconnect отключает автоматические соединения (правила для этой пары те же, что и для -addnode). Эта опция может быть указана несколько раз для подключения к нескольким узлам.

-обнаружить

Обнаружение собственных IP-адресов (по умолчанию: 1 при прослушивании, а не -externalip или -proxy)

-dns

Разрешить DNS-поиск для -addnode, -seednode и -connect (по умолчанию: 1)

-dnsseed

Запрашивать адреса одноранговых узлов через DNS-поиск, если адресов мало (по умолчанию: 1, если не используется -connection)

-enablebip61

Отправлять сообщения об отклонении по BIP61 (по умолчанию: 1)

-externalip = <ip>

Укажите свой публичный адрес

-forcednsseed

Всегда проверяйте адреса коллег через поиск DNS (по умолчанию: 0)

-listens

Принимать подключения извне (по умолчанию: 1, если нет -proxy или -connection)

-listenonionion

Автоматически создавать скрытый сервис Tor (по умолчанию: 1)

-maxconnections = <n>

Поддерживать максимум <n> соединений с пирами (по умолчанию: 125)

-maxreceivebuffer = <n>

Максимальный приемный буфер на соединение, <n> * 1000 байт (по умолчанию: 5000)

-maxsendbuffer = <n>

Максимальный размер буфера отправки на соединение, <n> * 1000 байт (по умолчанию: 1000)

-максимальная установка времени

Максимально допустимая поправка на поправку на среднее время пар. На эту местную перспективу может влиять одноранговое движение вперед или назад. (по умолчанию: 4200 секунд)

-maxuploadtarget = <n>

Попытайтесь сохранить исходящий трафик под заданной целью (в МиБ в течение 24 часов), 0 = без ограничений (по умолчанию: 0)

-onion = <> ip: порт>

Используйте отдельный прокси-сервер SOCKS5 для доступа к одноранговым узлам через скрытые сервисы Tor, установите -noonion для отключения (по умолчанию: -proxy)-onlynet=<net>

Создавайте исходящие соединения только через сеть <net> (ipv4, ipv6 или onion). Эта опция не влияет на входящие соединения. Эта опция может быть указана несколько раз, чтобы разрешить несколько сетей.

фильтры

Поддерживает фильтрацию блокировки и транзакции с фильтрами Блума (по умолчанию: 1)

-permitbaremultisig

Реле без P2SH multisig (по умолчанию: 1)

-port = <порт>

Прослушивайте соединения в <port> (по умолчанию: 8333, testnet: 18333, regtest: 18444)

-proxy = <ip: порт>

Подключитесь через SOCKS5 прокси, установите -poroxy для отключения (по умолчанию: отключено)

-proxyrandomize

Произведите рандомизацию учетных данных для каждого прокси-соединения. Это позволяет изолировать поток Tor (по умолчанию: 1)

-seednode = <ip>

Подключитесь к узлу для получения адресов одноранговых узлов и отключитесь. Эта опция может быть указана несколько раз для подключения к нескольким узлам.

-timeout = <n>

Укажите время ожидания соединения в миллисекундах (минимум: 1, по умолчанию: 5000)

-torcontrol = <ip>: <порт>

Порт управления Tor для использования, если включено прослушивание лука (по умолчанию: 127.0.0.1:9051)

-password = <пароль>

Пароль порта управления Tor (по умолчанию: пусто)

-upnp

Используйте UPnP для назначения порта прослушивания (по умолчанию: 0)

-whitebind = <addr>

Привязать к определенному адресу и пирам из белого списка, которые подключаются к нему. Используйте [host]: обозначение порта для IPv6

-whitelist = <IP или сетевой адрес>

Пары из белого списка подключаются с заданного IP-адреса (например, 1.2.3.4) или аннотированной сети CIDR (например, 1.2.3.0/24). Это может быть указано несколько раз. Белые пары не могут быть запрещены DoS

Варианты портфолио:

тип адреса

Какой тип адресов использовать («legacy», «p2sh-segwit» или «bech32», по умолчанию: «p2sh-segwit»)

избежать частичных расходов

Группируйте выходы по адресу, выбирая все или ни одного, а не выбирая по каждому выходу. Конфиденциальность улучшается, так как адрес используется только один раз (если кто-то не отправит его после того, как он его потратил), но может привести к несколько более высокой плате, поскольку выбор монет может быть неоптимальным из-за дополнительного ограничения (по умолчанию: 0)

тип замены

Какой тип изменений использовать ("legacy", "p2sh-segwit" или "bech32"). Значением по умолчанию является то же, что и -addresstype, за исключением случаев, когда -addresstype = p2sh-segwit выходной собственный segwit используется при отправке на собственный адрес segwit)

-из кошелька

Не загружайте кошелек и отключите вызовы RPC кошелька

-discardfee = <amt>

Ставка вознаграждения (в BTC / kB), указывающая на то, что вы можете отказаться от изменения, добавив его к вознаграждению (по умолчанию: 0,0001). Примечание. Выходные данные отбрасываются, если это пыль с такой скоростью, но мы всегда будем отбрасывать вплоть до скорости повторной передачи пыли и скорости выброса выше, которая ограничена оценкой скорости для самой длинной цели.

-fallbackfee = <amt>

Скорость тарифа (в BTC / kB), используемая, когда оценка скорости имеет недостаточно данных (по умолчанию: 0,0002)

-keypool = <n>

Установите размер пула ключей на <n> (по умолчанию: 1000)

-mintxfee = <amt>

Тарифы (в BTC / kB) меньше этой суммы рассматриваются как нулевая комиссия за создание транзакции (по умолчанию: 0,00001)

-paytxfee = <amt>

Плата (в BTC / kB) для добавления к отправляемым транзакциям (по умолчанию: 0,00)

-rescan

Пересканируйте блокчейн, чтобы найти пропущенные транзакции кошелька при запуске

-salvagewallet

Попробуйте восстановить приватные ключи из поврежденного кошелька при запуске

- проводить обмен информацией

Потратить неподтвержденные изменения при отправке транзакций (по умолчанию: 1)

-txconfirmtarget = <n>

Если ставка платежа не установлена, включите комиссию, достаточную для транзакций, чтобы начать подтверждение в среднем в течение n блоков (по умолчанию: 6)

-обслуживание портфеля

Обновить портфолио до последнего формата при запуске

-wallet = <путь>

Укажите путь к базе данных портфеля. Можно указать несколько раз для загрузки нескольких кошельков. Путь интерпретируется относительно <walletdir>, если он не является абсолютным, и будет создан, если он не существует (например, каталог, содержащий файл wallet.dat и registro). Для обратной совместимости он также примет имена существующих файлов данных в <walletdir>).

-walletbroadcast

Выполнять транзакции с портфельным спредом (по умолчанию: 1)

-walletdir = <dir>

Укажите каталог для сохранения кошельков (по умолчанию: <datadir> / wallets, если он существует, в противном случае <datadir>)

-walletnotify = <cmd>

Запустите команду при изменении транзакции кошелька (% s в cmd заменяется на TxID)

-walletrbf

Отправлять транзакции со всеми включенными опциями включения RBF (только RPC, по умолчанию: 0)

-zapwallettxes = <режим>

Очистить все транзакции кошелька и получить только части блокчейна через -scan при запуске (1 = сохранить метаданные tx, например, информацию о запросе платежа, 2 = метаданные drop tx)

Параметры уведомлений ZeroMQ:

-zmqpubhashblock = <адрес>

Включить блок публикации в <адрес>

-zmqpubhashblockhwm = <n>

Установите для блока отправки исходящих сообщений верхний водяной знак (по умолчанию: 1000)

-zmqpubhashtx = <адрес>

Разрешить публикацию транзакции гашиша в <адрес>

-zmqpubhashtxhwm = <n>

Установить верхний водяной знак сообщения о выходе после транзакции (по умолчанию: 1000)

-zmqpubrawblock = <адрес>

Включить необработанный блок публикации в <адрес>

-zmqpubrawblockhwm = <n>

Установить верхний водный знак для публикации необработанных сообщений (по умолчанию: 1000)

-zmqpubrawtx = <адрес>

Включить публикацию необработанной транзакции в <адрес>

-zmqpubrawtxhwm = <n>

Установить сообщение о выходе брутто транзакции после высокой отметки (по умолчанию: 1000)

Варианты отладки / тестирования:

-debug = <категория>

Выводить отладочную информацию (по умолчанию: -nodebug, указание <category> необязательно). Если <category> не указан или если <category> = 1, выводится вся информация отладки. <category> может быть: net, tor, mempool, http, bench, zmq, db, rpc, эстимейф, addrman, selectcoins, reindex, cmpctblock, rand, prune, proxy, mempoolrej, libevent, coindb, qt, leveldb.

-debugexclude = <категория>

Исключить отладочную информацию из категории. Может использоваться вместе с -debug = 1 для создания журналов отладки для всех категорий, кроме одной или нескольких указанных категорий.

-помощь-отладка

Распечатать справочное сообщение с параметрами отладки и выйти

-logs

Включить IP-адреса в выходные данные отладки (по умолчанию: 0)

-logtimestamps

Подготовьте выходные данные отладки с отметкой времени (по умолчанию: 1)-maxtxfee=<amt>

Максимальный общий сбор (в BTC) для использования в одной транзакции портфеля или транзакции брутто; установка слишком низкого значения может прервать большие транзакции (по умолчанию: 0.10)

-принт для консоли

Отправить информацию трассировки / отладки на консоль (по умолчанию: 1, когда нет -daemon. Чтобы отключить ведение журнала в файл, установите -nodebuglogfile)

-shrinkdebugfile

Сжатие файла debug.log при запуске клиента (по умолчанию: 1, когда нет -debug)

-uacomment = <cmt>

Добавить комментарий в цепочку пользовательских агентов

Варианты выбора цепи:

-testnet

Используйте тестовую цепочку ...

Опции реле узла:

-bytespersigop

Эквивалентные байты на сигоп в транзакциях повторной передачи и майнинга (по умолчанию: 20)

-datacarrier

Мои и ретранслирующие транзакции на носителях данных (по умолчанию: 1)

-datacarrierize

Максимальный размер данных в транзакциях носителей данных, которые мы передаем и извлекаем (по умолчанию: 83)

-metropool-замена

Включить подстановку транзакций в пуле памяти (по умолчанию: 1)

-minrelaytxfee = <amt>

Комиссионные (в BTC / КБ) меньше этой суммы считаются нулевой комиссией за повторную передачу, снятие и создание транзакций (по умолчанию: 0,00001)

-whitelistforcerelay

Принудительная повторная передача одноранговых транзакций из белого списка, даже если транзакции уже были в metropool или нарушают локальную политику повторной передачи (по умолчанию: 0)

-whitelistrelay

Принимать передаваемые транзакции, полученные от узлов из белого списка, даже если транзакции не передаются (по умолчанию: 1)

Варианты создания блока:

-blockmaxweight = <n>

Установите максимальный вес блока BIP141 (по умолчанию: 3996000)

-blockmintxfee = <amt>

Установите самую низкую комиссию (в BTC / КБ) для транзакций, которые включены в создание блоков. (по умолчанию: 0,00001)

Параметры сервера RPC:

-Ресторан

Принимать публичные REST-запросы (по умолчанию: 0)

-rpcallowip = <ip>

Разрешить соединения JSON-RPC из указанного источника. Они действительны для <ip> одного IP (например, 1.2.3.4), сети / сетевой маски (например, 1.2.3.4/255.255.255.0) или сети / CIDR (например, 1.2.3.4/24).). Эта опция может быть указана несколько раз

-rpcauth = <userpw>

Имя пользователя и пароль HMAC-SHA-256 для соединений JSON-RPC. Поле <userpw> имеет формат: <имя пользователя>: <SALT> \$ <HASH>. Канонический скрипт на python включен в share / rpcauth. Затем клиент обычно подключается, используя пару аргументов rpcuser = <USERNAME> / rpcpassword = <PASSWORD>. Эта опция может быть указана несколько раз

-rpcbind = <addr> [: порт]

Привязать к определенному адресу для прослушивания соединений JSON-RPC. Не подвергайте сервер RPC ненадежным сетям, таким как общедоступный Интернет! Эта опция игнорируется, если также не передана опция -rpcallowip. Порт является необязательным и переопределяет -rpcport. Используйте [host]: обозначение порта

для IPv6. Эта опция может быть указана несколько раз (по умолчанию: 127.0.0.1 и ::1, т.е. localhost)

-rpccookiefile = <loc>

Расположение авторизационного куки. Относительные маршруты будут иметь префикс определенного местоположения сетевого каталога данных. (по умолчанию: data dir)

-rpcpassword = <pw>

Пароль для соединений JSON-RPC

-rpcport = <порт>

Прослушивать JSON-RPC-соединения на <порт> (по умолчанию: 8332, testnet: 18332, regtest: 18443)

-rpcserialversion

Установите сериализацию необработанной транзакции или возвращенного шестнадцатеричного блока в режим non-verbose, non-segwit (0) или segwit (1) (по умолчанию: 1)

-rpcthreads = <n>

Установите количество потоков для обслуживания вызовов RPC (по умолчанию: 4)

-rpcuser = <пользователь>

Имя пользователя для соединений JSON-RPC

-server

Принять командную строку и команды JSON-RPC

32. Лицензирование и использование программного обеспечения.

Android

<https://source.android.com/setup/start/licenses>

Termux

<https://github.com/termux/termux-app/blob/master/LICENSE.md>

Node

<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-rest

<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>

Tor Network

<https://github.com/torproject/tor/blob/master/LICENSE>

Syncthing Network

<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 Companion y App Inventor Blockly

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

SQLite Expert Personal -freeware

<http://www.sqliteexpert.com/download.html>

Apache Ant

<https://ant.apache.org/license.html>

WGET

<https://www.gnu.org/software/wget/>

OpenJDK

<https://openjdk.java.net/legal/>

Extensiones externas:

JSONTOOLs

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

Лицензирование OpenSource и коммерческих версий системы Mini BlocklyChain можно найти на официальном сайте <http://www.openqbit.com>.

Mini BlocklyChain, MiniBlockly, BlocklyCode, MiniBlockMiniChain, QBlockly являются зарегистрированными товарными знаками OpenQbit.

Мини BlocklyChain находится в свободном доступе.

Весь код и документация в Mini BlocklyChain была передана авторам в общественное достояние. Все авторы кода и представители компаний, в которых они работают, подписали аффидевиты, посвящающие их вклад в общественное достояние, а оригиналы этих аффидевитов хранятся в сейфе в главных офисах OpenQbit México. Любой может свободно публиковать, использовать или распространять оригинальные расширения Mini BlocklyChain (OpenQbit), либо в виде исходного кода, либо в виде скомпилированного двоичного файла, для любых целей, коммерческих или некоммерческих, а также любыми средствами.

Предыдущий абзац относится к поставляемому коду и документации в Mini BlocklyChain, тех частях библиотеки Mini BlocklyChain, которые фактически объединяются и поставляются с более крупным приложением. Некоторые сценарии, используемые как часть процесса сборки (например, сценарии «конфигурации», сгенерированные autoconf), могут быть включены в другие лицензии с открытым исходным кодом. Однако ни один из этих сценариев сборки не попадает в окончательную поставляемую библиотеку Mini BlocklyChain, поэтому лицензии, связанные с этими сценариями, не должны учитываться при оценке ваших прав на копирование и использование библиотеки Mini BlocklyChain.

Весь поставляемый код в Mini BlocklyChain был написан с нуля. Ни один код не был взят из других проектов или из открытого Интернета. Каждая строка кода может быть прослежена до ее первоначального автора, и все эти авторы имеют открытые области доступа к файлам. Таким образом, база кода Mini BlocklyChain является чистой и не загрязнена кодом, лицензированным из других проектов с открытым исходным кодом, а не открытым вкладом.

Mini BlocklyChain имеет открытый исходный код, что означает, что вы можете делать столько копий, сколько хотите, и делать то, что вы хотите с этими копиями, без ограничений. Но Mini BlocklyChain не является открытым. Чтобы сохранить Mini BlocklyChain в открытом доступе и убедиться, что код не загрязнен проприетарным или лицензионным контентом, проект не принимает патчи от неизвестных людей. Весь код в Mini BlocklyChain является оригинальным, так как он был написан специально для использования Mini BlocklyChain. Ни один код не был скопирован из неизвестных источников в Интернете.

Mini BlocklyChain находится в открытом доступе и не требует лицензии. Тем не менее, некоторые организации хотят получить юридическое подтверждение вашего права на использование Mini BlocklyChain. Обстоятельства, при которых это происходит, включают следующее:

- Ваша компания требует компенсации за претензии о нарушении авторских прав.
- Вы используете Mini BlocklyChain в юрисдикции, которая не признает общественное достояние.
- Вы используете Mini BlocklyChain в юрисдикции, в которой не признается право автора передавать свои работы в открытый доступ.
- Вы хотите иметь материальный юридический документ в качестве доказательства того, что у вас есть законное право использовать и распространять Mini BlocklyChain.
- Ваш юридический отдел говорит вам, чтобы купить лицензию.

Если что-то из вышеперечисленного относится к вам, OpenQbit, компания, в которой работают все разработчики Mini BlocklyChain, продаст вам гарантию прав собственности на Mini BlocklyChain. Гарантия права собственности - это юридический документ, в котором говорится, что заявленные авторы Mini BlocklyChain являются настоящими авторами, и что авторы имеют законное право передать Mini BlocklyChain в открытый доступ и что OpenQbit будет энергично защищать себя от претензий по лицензированию. Все доходы от продажи титульных гарантий Mini BlocklyChain используются для финансирования постоянного улучшения и поддержки Mini BlocklyChain.

Введенный код

Чтобы сохранить Mini BlocklyChain полностью свободным и защищенным авторским правом, проект не принимает патчи. Если вы хотите внести предлагаемое изменение и включить патч в качестве доказательства концепции, это было бы здорово. Однако не обижайтесь, если мы переписаем ваш патч с нуля. Тип некоммерческого лицензирования или лицензирования с открытым исходным кодом, который использует его в этом режиме и некоторые аналогичные без приобретения поддержки, индивидуального или корпоративного использования, независимо от размера компании, регулируется следующими правовыми условиями.

Отказ от гарантии. Если это не требуется действующим законодательством или не согласовано в письменной форме, Лицензиар предоставляет Работу (а каждый Участник предоставляет Вклады) «КАК ЕСТЬ», БЕЗ ГАРАНТИЙ ИЛИ УСЛОВИЙ ЛЮБОГО РОДА, явных или подразумеваемых, в том числе, без ограничений, любая гарантия или условие НАЗВАНИЯ, НИКАКИХ НАРУШЕНИЙ, ТОВАРНОЙ ИЛИ ПРИГОДНОСТИ ДЛЯ ОСОБЫХ ЦЕЛЕЙ. Вы несете единоличную ответственность за определение правильного использования или перераспределения Работы и принятие на себя любого риска,

связанного с использованием вами разрешений в соответствии с настоящей Лицензией.

Любые финансовые потери или потери любого рода из-за использования этого программного обеспечения не будут иметь возможности оплаты любого вида. Все судебные споры сторон будут переданы в суды только в юрисдикции Мехико, страны Мехико.

Для поддержки, использования и коммерческого лицензирования должно быть заключено соглашение или договор, заключенный между OpenQbit или его корпорацией и заинтересованной стороной.

Условия распространения рекламы могут быть изменены без предварительного уведомления. Посетите официальный портал www.openqbit.com, чтобы узнать о любых изменениях положений о некоммерческой и коммерческой поддержке и лицензировании.

Любое лицо, пользователь, частное лицо, публикующее материалы любой юридической природы или из любой части мира, которые просто используют программное обеспечение, безусловно, принимает положения, установленные в этом документе, и те, которые могут быть изменены в любое время на портале www.openqbit.com без предварительного уведомления и может применяться по усмотрению OpenQbit для некоммерческого или коммерческого использования.

Любые вопросы и информацию о Mini BlocklyChain следует адресовать сообществу App Inventor или сообществам различных систем Blockly, таких как: AppBuilder, Trunkable и т. д. и / или по адресу opensource@openqbit.com по запросу, для ответа может потребоваться от 3 до 5 рабочих дней

Поддержка с коммерческим использованием.

support@openqbit.com

Продажи коммерческого использования.

sales@openqbit.com

Правовая информация и вопросы лицензирования или проблемы.

legal@openqbit.com

