# INTRODUCTION

*Give a brief presentation of the coin. You can put a link to a general article, e.g. blog article*
*What is the goal of the project? What makes it different from other projects?*
*List its main features.*

Polkadot is a project aiming at making blockchains of all kinds able to communicate with each other. It achieves this via a main *Relay Chain* acting as a hub interfacing other blockchains.
It is based on Substrate, a blockchain toolkit which provides the NPoS consensus and the transaction management, among other things.
It is very close to Cosmos, both technically and conceptually.

Polkadot key features are:

- Fast-finality blockchain: 1 **valid** block every ~6 seconds
- Nominated Proof-of-Stake (NPoS) consensus
- Staking with validators
- Rewards

# GLOSSARY

*Most coins use their own terms and definitions. Identify and give a description for any of those terms that will be used in this coin integration in Ledger Live.*

See [DOT] - Glossary

# ECOSYSTEM

## Explorers

*List all the existing blockchain explorers*

- https://polkascan.io/
- https://polkadot.polkastats.io/
- https://polkadot.subscan.io/
- https://polkadot.js.org/apps/#/explorer

## Indexer

*Does the node have native history capabilities? I.e. does it natively provide a way to query all past transactions of an account since genesis?*
*In certain cases, nor the standard node neither existing explorers easily provide transactions history.*
*To determine if an additional indexer is required,* ***please refer to the indexer section of the developer portal****.*

The Polkadot node has no native indexing capability, we need to either find a 3rd-party provider or develop it internally.

Indexer specification, defined by us: [DOT] - Indexer API

Existing indexer projects (none of which suits our needs):

- Polkascan : https://github.com/polkascan/polkascan-os
- Polkastats : https://github.com/Colm3na/polkastats-backend-v3
- Subscan : https://github.com/itering/subscan-essentials
- PSQL indexer project : https://github.com/usetech-llc/polkadot_psql_indexer

## Running node providers

*List potential 3rd party partners able to deploy and run the node.*

**BisonTrails** seems to be the only running node provider with a commercial offer so far: https://bisontrails.co/polkadot/
They may provide us with an indexer as well (**TO BE CONFIRMED**).

## Wallets

*Is there an official wallet (maintained by the coin's developers)?*
*Are there 3rd-party wallets?*
*For each of them: what derivation paths does it support? Is it compatible with the Ledger Nano? Provide location of the source code if available.*

- Main wallet : https://polkadot.js.org/apps/ (Google Chrome only)
- Other wallets : https://wiki.polkadot.network/docs/en/build-wallets#supported-wallets (most of them don't have Ledger Nano support)

# NODE

*Is there a running MainNet? Are there TestNets? DevNets?*
*Is the coin part of a broader family (eg. many coins are based on Bitcoin, Polkadot is based on Substrate, etc...)?*
*Briefly describe how to run a node locally (useful for early/quick testing), or provide a link describing the procedure.*

## Implementation

Source code main repository: https://github.com/paritytech/polkadot/

Polkadot is largely based on Substrate, which is coded in Rust. As a multipurpose blockchain framework, Substrate provides many modules, called pallets, each of which provide a blockchain functionality.
Of the great number of existing pallets, probably only a few are relevant for us:

- pallet_balances
- pallet_identity
- pallet_staking
- pallet_transaction_payment[_rpc].

The full list of existing pallets can be found here (pallet_balances as an example).

## Installation

Linux users can retrieve precompiled binaries from here: https://github.com/paritytech/polkadot/releases
Other systems can either:

- run it in Docker: https://github.com/paritytech/polkadot/blob/master/doc/docker.md
- install it with cargo: cargo install --git https://github.com/paritytech/polkadot --tag <version> polkadot --locked
- build it from source code in Rust: https://github.com/paritytech/polkadot/#build-from-source

## Configuration

A few relevant parameters to provide when starting the node:

**--name "your node's name"**: once started, info about the node can be found at https://telemetry.polkadot.io/#list/Polkadot
**--pruning archive**: ensures the node has access to the complete blockchain, not only the last blocks
**--wasm-execution Compiled|Interpreted**: you should use Compiled when starting the node for the 1st time, makes synchronization way faster but more resource consuming. Once synchronized you should stop the node and restart it with Interpreted to use less resources.
**--rpc-external**: enables distant access to the node via HTTP/RPC
**--rpc-cors**: allow some domains/ip to access the node via RPC or WS - value "all" disables the blocking mechanism
**--ws-external**: enables remote access to the node via WebSocket

## API

*What are the available techniques to get and send info from/to the node?*
*What endpoints does the node provide?*
*Provide a link to it's API documentation, if available.*
*Are there available wrappers on top of the raw node (eg. a library to expose a REST API on top of a default RPC interface)? Document them too.*

The native node provides an RPC API, available through either **HTTP** or **WebSocket**.
It can be retrieved from the node itself using this call (replacing localhost by any relevant node URL):

```
curl -H "Content-Type: application/json" -d '{"id":1, "jsonrpc":"2.0", "method": "rpc_methods"}' http://localhost:9933/
```

*Note: most of the Substrate stored data are encoded in SCALE format, making almost mandatory the use of up-to-date protocol libraries.*

# CRYPTOGRAPHY

### HD Wallet Derivation path

*What is/are the derivation path(s) supported by the Nano app?*
*Is it BIP44? BIP-32?*
*Is it account-based? UTXO-based?*

Supported by the Nano app: m/44'/354'/account'/change'/index'
Supported by the *polkadot.js* official wallet using Nano: m/44'/354'/account'/0'/index' (to the user's choice)
It is likely that we support only the account derivation in Ledger Live, since polkadot.js' app derivation path doesn't make much sense.

### Elliptic curves

*How are the transactions signed?*
*What are the curves used?*
*Are there specific algorithms used?*

All substrate-based blockchains support the following curves:

- Edwards (ed25519)
- Schnorrkel (sr25519)
- secp256k1

The Polkadot Nano app only supports **ed25519** to date, support of sr25519 is planned but no ETA.

### Address

*How are addresses constructed starting from the public key?*

All substrate-based blockchains use SS58 for pubkey-to-address conversion: base58encode ( concat ( <pubkey-type>, <pubkey>, <checksum> ) )
A same public key can be used on any substrate-based blockchain but will be formatted differently on each, resulting on a different address, due to the use of a network-specific prefix.
Polkadot addresses always begin with **1** e.g: 12JHbw1vnXxqsD6U5yA3u9Kqvp9A7Zi3qM2rhAreZqP5zUmS (0/0/0 address for Obelix seed).

## CURRENCY

*What is the main currency of the coin?*
*How many decimals does it support?*
*List all existing units.*
*What is the total supply? How many are currently circulating?*

The main Polkadot currency is the DOT, which smallest unit is the PLANCK where 1 DOT = 10,000,000,000 PLANCK.
The total supply of DOT is about 1,000,000,000 as of September 2020.

Note: Polkadot originally launched with a 12 decimals precision, and then switched to a 10 decimals precision following the DOT redenomination that took place on 21 August 2020, now to be known as Denomination Day, at block #1,248,328. The previous currency is referred to as "**DOT (old)**" which is now 100 DOT. It is only a cosmetic change of the currency name, the network still works with the same constant value for a PLANCK.

### Tokens

*Does the coin support tokens, other than the native currency?*
*Detail all token operations to be supported on Ledger Live: Creation? Transfer? Deletion? Any other?*

The Polkadot Relay Chain doesn't support tokens, apart from its native DOT currency.
However parachains, as any substrate-based blockchain, can support their own main currency as well as custom tokens.
Only the Relay Chain will be supported on Ledger Live integration.

## ECONOMICS

### Balances

*List and explain all types of balances that describe the user's funds (available, locked, allocated to staking or voting, etc...)*

Besides the "total" balance, there are several types of balances associated to a Polkadot account:

- **Free balance**: the balance that can be used at any moment, e.g for a transfer or paying fees.
- **Bonded balance**: the balance that is currently locked for staking. Bonding takes 2 eras to be effective.
- **Unbonding balance**: the balance that is bonded and waiting to be unlocked. The unbonding period is 28 eras on Polkadot.
- **Locked balance**: a balance that has rules and restrictions on its use.
- **Vesting balance**: an example of locked balance, where the balance has a lock that decreases over time.

#### Minimum balance

*Is there an account creation cost?*
*Is there an account minimum balance? Is it possible to go below, and if so, what happens?*
*How can an account be emptied?*

*Is there a minimum amount for staking?*

An account on Polkadot has a minimum required balance (called the *Existential Deposit*) of **1 DOT**.

If the total balance of the account goes below 1 DOT for any reason, the account is *reaped*, which means all its remaining funds (less than 1 DOT) are burnt, putting its balance at 0, and its nonce is reset to 0.
When reaped an account has no current state on the blockchain, however, its full operation history remains available and it can be reused, by transferring 1 DOT or more to it.
The blockchain provides a *transfer keep-alive* transaction type, that prevents the account to be reaped by making the transaction fail instead.

All the types of balance are taken into account for the minimum balance, which typically means if the accounts have 1 DOT or more of *bonded balance*, its *free balance* can be 0 DOT and therefore it can't be reaped by a transfer.

Using the polkadot.js/api library, relevant balances can be retrieved using api.derive.balances.all(address) and/or api.query.balances.account(address) and/or api.query.staking.ledger(address).

**Spendable balance**

*Ledger Live defines the spendable balance as the maximum amount the user can send to another address, immediately and without any restriction.*
*Describe the formula to compute that spendable balance, taking into account all rules, constraints, fees, types of balances…*

For a Polkadot transfer transaction, the formula for the spendable balance should be:

```
spendable = free - (1 - min(1, bonded)) - fees
```

## Rewards

*Is there any way to receive rewards using this coin? Describe how.*
*What conditions must be met to receive rewards?*
*Describe how the amount of rewards is calculated or provide a link to an explanation.*

The rewards computation mechanism is explained here:
https://wiki.polkadot.network/docs/en/maintain-guides-validator-payout#nominators-and-validator-payments

Rewards generation process:

1. Rewards are generated every era (24h on Polkadot), for all *elected* validators of that era.
2. At the end of each era, the total amount of generated rewards is split **evenly** between all *elected* validators, **regardless of their stake**.
3. Each validator takes its commission, an arbitrary percentage that can be from 0% to 100%
4. What's left is split between the nominators for which the validator is *active* for that era, **proportionally to their stake**.

The algorithm to retrieve a nominator's rewards should look like:

```
1) Get the total rewards for the current era
2) Get the number of elected validators for that era (changes via governance, not often)
3) Compute rewards for any validator = total rewards / nb elected validators
4) Compute rewards for 1 nominator of the validator = (validator's reward - validator's commission) * (nominator's stake / total validator's stake)
```

## Fees

*Are fees estimated or exact?*
*Are they chosen by the user or set by the blockchain?*
*What is a typical transaction cost?*
*Do failed transaction charge fees?*
*Mention any special case, e.g. for grouped transactions…*

Unlike other blockchains, on Polkadot fees are not part of the transaction. They are managed as an event, i.e a "side effect" of a transaction. This has the following consequences:

- Fees are not displayed on the Nano app during signature **To be approved during Nano app review**
- A failed transaction does not always pay fees, depending on the situation:
  - If the transaction is a transfer, and the account's free balance is less than the transferred amount but enough to pay the fees → the fees are paid and the tx is recorded as failed on the blockchain.
  - For other types of transactions, if the account's free balance is less than the fees → nothing is paid, and the transaction is not recorded on the blockchain.

Estimated fees can be retrieved directly from the node with an RPC call to payment_queryInfo, of following this example with polkadot.js/api.

## Gas

*Do transactions need gas? If so how does it work?*

## Slashing

*Is there any risk of being slashed using this coin? Describe how.*

Validators will be slashed if they misbehave, i.e do not comply with required duties. When slashing happens, both the validator and all of its current nominators lose a percentage of their funds that depends on the offense.
More info here: https://wiki.polkadot.network/docs/en/learn-staking/#slashing

### Constants

*Blockchains often define many constants (whose value may sometimes change, despite their name…) that contribute to the overall behavior of the network.*
*List and describe any constant relevant to this coin integration.*
*Provide the constant's value at the time of writing, with source if possible.*

Value of some constants of the system, **at the time of writing**. Any of these constants can change following a governance vote.

- Existential deposit = **1 DOT** (consts.balances.existentialDeposit - source)
- Minimum bonded balance = **1 DOT** (source?)
- Duration of a block (expected) = **6 seconds** (consts.babe.expectedBlockTime)
- Duration of a session = **4 hours - 2400 blocks** (consts.babe.epochDuration)
- Sessions per era = **6** (consts.staking.sessionsPerEra)
- Duration of an era = **24 hours - 14400 blocks** (sessionsPerEra * epochDuration)
- Maximum number of unbonding requests at a same time : **32** (source)
- Maximum number of nominators who get rewards per validator/era, validator is *oversubscribed* if above: **128**consts.staking.maxNominatorRewardedPerValidator - source)
- Maximum number of validators a nominator can elect during a single era : **16** (source)
- Bonding duration (unbonding delay) = **28 eras** (consts.staking.bondingDuration - source)
- Number of elected validators per era (ideal) = **204** (api.query.staking.validatorCount)

The polkadot.js wallet provides a developer webapp to fetch these constants and others : https://polkadot.js.org/apps/#/chainstate

## BLOCK STRUCTURE

*Describe the information contained in one block, or provide a link to a representative block example.*

⚠ ***Make sure that blocks contain a timestamp**, this is mandatory for Ledger Live's history features (operations list and balance history).*

Blocks are identified by their height since the genesis block, e.g 1835821.
Blocks can contain extrinsics, events and logs.

- Extrinsics can be either transactions or *inherents*. Inherents are individual actions independent from a transaction, a classic example is the assignment of a timestamp to the block.
- Events carry a lot of meaningful information on top of transactions. For example, fees and rewards amounts.
- Logs carry transverse information and metadata about different modules of the blockchain.

To gather all the information we need we will have to consider both transactions, inherents, events and maybe logs. This will most likely be done by the indexer.

All these elements are identified by their index in the block, and follow the same identification scheme but independently from each other. For example, 1835821-0 references both the first extrinsic, the first event and the first log of block 1835821.

Explorer example of raw block data: https://pastebin.pl/view/raw/e8996c93

## TRANSACTIONS

### Tx broadcast format

*What is the serialization format used when broadcasting a signed transaction? (JSON, MsgPack, SCALE…)*
*Are there available libraries to support encoding/decoding to that format?*

Transactions, as most of the data on the blockchain, are encoded in SCALE format.
We can rely on the polkadot.js/api library to handle the encoding/decoding.

### Tx format as received from the network

*In what format are the transactions provided when queried from the blockchain? (JSON, MsgPack, SCALE…)*
*Are there available libraries to support encoding/decoding to that format?*

Polkadot does not provide a direct way to fetch transactions from an account address.
Then, we need to use an indexer (which will scan and index all the blocks and their content into a dedicated database) to aggregate some data and fetch the input and output operations of an account.

**TODO**: depends on indexer spec, see https://ledgerhq.atlassian.net/wiki/spaces/LCH/pages/2214494335

## Tx types

*List and describe all the transaction types that are relevant for this integration in Ledger Live.*

There are a lot of transaction types in Polkadot, these are those we are interested in, as a wallet provider:

- transfer: a regular send. Subject to account reaping if it causes the account to go below the existential deposit of 1 DOT
- transferKeepAlive: same as transfer but protects against reaping by making the transaction fail instead
- bond: stake some funds
- bondExtra: change the current amount of stake
- unbond: declare the intention to unstake some funds: these funds become available after 28 eras
- rebond: declare the intention to stake some funds that are unbonding (but not yet unlocked)
- withdrawUnbonded: retrieve the funds when they become available after the 28 eras unbonding period **(NOT TESTED YET)**
- nominate: define the set of validators to elect for staking (same transaction type for first assignment and for replacing them)
- chill: declare the desire to cease validating or nominating. Does not unbond funds.
- payoutStakers: trigger the retrieval of pending rewards for a given nomination, to the validator and its 256 (currently, the value may change) top staking nominators

## Tx fields

*What are the fields and structure of an unserialized transaction?*
*Do transactions require a sequence number (nonce)?*

**Common to all transactions**

- address: The SS58-encoded address of the sending account.
- blockHash: The hash of the block.
- blockNumber: The height of the block since genesis.
- genesisHash: The genesis hash of the chain.
- metadataRpc: The SCALE-encoded metadata for the runtime when submitted.
- nonce: The nonce for this transaction - an incremental value associated to the account.
- specVersion: The current spec version for the runtime.
- transactionVersion: The current version for transaction format.
- tip: [optional] The tip to increase transaction priority.
- eraPeriod: [optional] Describe the longevity of a transaction. It represents the validity from the blockHash field, in number of blocks. Defaults to 64 blocks, If 0 the transaction is immortal.
- validityPeriod: [optional] The amount of time (in seconds) the transaction is valid for. Defaults to 5 minutes, if 0 the transaction is immortal.

**Transfer / transfer keep-alive (doc)**

- dest: The destination address
- value: The amount of DOT to be transferred

**Bond (doc)**

Must be signed by a Stash account.

- controller: The controller address for this bond
- payee: The rewards destination. Can be "Stash", "Staked", "Controller" or any account address.
- value: The amount of DOT to be bonded

**BondExtra (doc)**

Must be signed by a Stash account.

- maxAdditional: The DOT amount to add to the current bond.

**Unbond (doc)**

Must be signed by a Controller account.

- value: The amount of DOT to unbond.

**Rebond (doc)**

Must be signed by a Controller account.

- value: The amount of DOT to rebond.

**WithdrawUnbonded (doc)**

Must be signed by a Controller account.

- numSlashingSpans

**Nominate (doc)**

Must be signed by a Controller account.

- targets: The array of addresses of the targets to nominate (maximum 16). These addresses replace any previous ones. Provides no checks as to whether these targets are actual validators.

**PayoutStakers (doc)**

- era: May be any era between [current_era - history_depth; current_era]. Substrate only retains up to history_depth eras of reward information.
- validatorStash: The Stash account of a *validator*. Their nominators, up to the maximum MaxNominatorRewardedPerValidator, will also receive their rewards.

## CONSENSUS

*Describe how the network achieves consensus, i.e. how all participating nodes agree on the next valid block to add to the blockchain.*
*Is the consensus mechanism PoW or PoS? What flavor (DPoS, NPoS, PPoS...)?*
*Does it involve validators? How?*
*How long does it take to create a valid block?*
*Is a blockchain reorg possible?*

Polkadot's Nominated Proof-of-Stake consensus, provided by substrate, is very similar to Cosmos DPoS. The system has validators who are in charge of creating valid blocks with a requirement for stability and continuity of service. The system also has nominators, who are users with no desire to be validators, but still wish to participate in the consensus process. Users can also chose to neither validate nor nominate, in which case they *chill*.

Polkadot's consensus algorithm is the Sequential Phragmén Method, a multi-winner election method introduced by Edvard Phragmén in the 1890s. This method is used in the NPoS scheme to elect validators based on their own self-stake and the stake that is assigned to them from nominators. It also tries to equalize the weights between the validators after each election round. Since validators are paid equally in Polkadot, it is important that the stake behind each validator is spread out. Phragmén tries to optimize three metrics in its elections of validators:

1. Maximize the total amount at stake.
2. Maximize the stake behind the minimally staked validator.
3. Minimize the variance of the stake in the set.

Improved Phragmén's method is a time consuming procedure. In order to keep a constant block time of six seconds, Polkadot uses off-chain workers to perform the election and then submit a transaction to propose the set of winners.
Because certain user actions, like changing nominations, can change the outcome of the Phragmén election, the system forbids calls to these functions for the last **25%** of the session before an era change.
During that period, these transaction types are not permitted:

- bondExtra
- unbond
- rebond
- withdrawUnbonded
- validate
- nominate
- chill
- payoutStakers

## FEATURES

*Some features of the coin do not need to be analysed in detail here, as they are usually not included in Ledger Live integrations.*
*These typically include **Governance** and **Smart contracts**, for example.*

### STAKING

*Describe all staking user flows. Feel free to provide flowcharts.*
*Describe staking-related concepts that are specific to the coin.*
*Is there a waiting time between a request to unlock funds and the funds being available? (e.g. 21 days for Cosmos, 28 days for Polkadot).*

Staking in Polkadot is very similar to Cosmos from a UX perspective: regular users can chose to bond some funds and elect some validators, which will generate rewards or slashing.
Unlike Cosmos however, bonding and nominating are two separate actions: modifying the bonded balance and the chosen validators can be done independently, in separate transactions. The reason behind this is that in Polkadot **the user does not chose the amount to assign to each validator**, instead of this his total bonded funds are automatically split by the consensus algorithm between all of his *active* nominations (see below).

The blockchain allows to manage bonded funds and nominations in a same transaction using a utility.batch extrinsic, but this is not yet supported by the Nano app, and it is just an optional convenience.

#### Validating

The account directly participates in the NPoS consensus by validating transactions and creating blocks. This requires to run a node and to abide by with some duties.
Validators can be in 2 states:

- **waiting**: the validator is in the general pool of all existing validators, waiting to be elected to participate in consensus. Validators in this state receive no rewards.
- **elected**: the validator has been selected to participate in consensus, and receive rewards for doing so.

#### Nominating

The account does not directly participate in the transaction validation and block creation process, however it bonds some of its funds and selects a set of validators (up to 16).
Bonding some funds and nominating validators can be done independently via separate transactions. At any moment, there must be bonded funds to be able to have nominations. Besides this constraint everything is independent: changing the bonded amount, changing the list of nominated validators, stop from nominating, can all be done independently.

For an *elected* validator, a nomination can be in 2 states:

- **inactive**: the validator is elected, but the Phragmén algorithm hasn't assigned it to the nominator (see *Consensus*). The nominator will not receive rewards from this validator.
- **active**: the validator is elected and the Phragmén algorithm has assigned it to the nominator, who will therefore receive rewards from it.

#### Chilling

Neither nominating nor validating. Default state of a new account. Independent of the bonding status: there can be bonded funds while chilling.

#### Controller & Stash

When nominating, Polkadot allows to define 2 different accounts with separate roles:

- **Stash account**: this is the primary account that holds the bonded funds. It is used to sign transactions that increase bonded funds: `bond` and `bondExtra`.
- **Controller account**: this account is allowed to control all other staking operations on behalf of the stash account: `unbond`,`rebond`, `withdrawUnbonded`, `nominate`, `chill`, `payoutStakers`. The controller account only needs enough funds to pay the fees of the transactions it makes.

The point of this Polkadot feature is to enhance security by allowing users to leave their stash account in a cold wallet for most operations.

Since the Nano is a cold wallet, there is no security issue in using the same account for stash and controller. This will most likely be the preferred functionality in Ledger Live.

#### Proxy accounts

Polkadot provides a feature similar but more generic than the stash/controller relationship, where an account can be a proxy of another account, with different proxy modes allowing different types of operations:

- Any
- Non-transfer
- Governance
- Staking
- Identity Judgement

A deposit is required to create proxy accounts, which value on Polkadot is `20.008 + 0.033 * nb proxies`

Here is an example of how a main account can benefit from proxy, stash and controller accounts:

### GOVERNANCE

On-chain governance, similar to Cosmos.

### SMART CONTRACTS

There are is no support of smart contracts on the Polkadot Relay Chain.

### OTHER FEATURES

*Does the coin have other high-level features worth mentioning?*

## NANO APPLICATION

*What are the Nano app interfaces and APDUs? Mention at least those for scanning accounts, reading addresses and signing transactions.*
*Is there an available JS library to discuss with the Nano app? Provide location of the source code if available.*

# DEVELOPER RESOURCES

### Communication channels

*Identify communication channel(s) with the devs of the blockchain.*
*Identify communication channel(s) with the devs of the Nano app.*
*(chat app channel, forum, support platform, mailing list…?)*

### Documentation

*List all official and/or unofficial documentation resources*

### JS libraries

*Sometimes SDKs can provide some parts of the implementation and can be used directly in Ledger Live.*
*For instance there may be libraries for address serialization, or to communicate with the Nano app, or with the explorer... List any such resource here.*

- Many JS libs here: https://github.com/polkadot-js
- Transactions: https://github.com/paritytech/txwrapper
- Nano app JS interface: https://github.com/Zondax/ledger-polkadot-js

# APPENDIX: JAVASCRIPT LIBRARY USAGE

*Provide examples, or minimal implementations, of how 3rd party JS libraries can be used in the context of this coin integration.*

Polkadot's host can be operated with a HTTP or WebSocket RPC API client.
Some of the features can be accessed encoding/decoding the SCALE format. Most of them require using subscriptions through WebSocket (that's why HTTP RPC methods and features are limited).

Using the polkadot.js/api client library is the best way to interact with the polkadot host. It contains multiple packages including (but not limited to):

- api: the client library itself with ApiPromise and ApiRx
- types: the substrate types used in the library. NB: it's not only the typescript definitions, but all the classes and their implementations...
- rpc-core: the generic RPC client
- rpc-provider: the transport for the api client. We can use WsProvider for WebSocket or HttpProvider for a limited usage of RPC
- api-derive: a collection of common functions of Polkadot, using multiple api calls and aggregating data together for ease.

*This library is a Substrate client library, not dedicated to Polkadot. It adapts dynamically to the metadata provided by the runtime. If some features are not handled by the Substrate node it connects to, it won't have some methods registered. Thus, it is good practice to always check for method existence before calling them, depending on the pallets of the Substrate node.*

We will detail here some of the typical api calls we would use through examples.

*NOTE: some code parts use the* at(blockHash) *or* keys() *methods. Those are respectively used to query some data at a specified block hash, or get the list of keys from maps. These advanced usage are described in Polkadot.js documentation - Query extras. You can replace* method.at(blockHash, ...arg) *by* method(...arg) *to get data from latest block.*

### Creating the client

Polkadot JS API is best used through WebSocket, because it enables the full features through subscriptions.
We can choose between a Promise api, or a RxJS api, but here is an example using the ApiPromise:

```
import { ApiPromise } from '@polkadot/api';
import { WsProvider } from '@polkadot/rpc-provider';

const WS_URL = "ws://localhost:9944";
const WS_AUTHORIZATION = "Basic base64_encoded_user:password"; // Not standard but works for BisonTrails authorization

async function createApi() {
  const wsProvider = new WsProvider(WS_URL, { Authorization: WS_AUTHORIZATION });
  const api = await new ApiPromise({ provider: wsProvider }).isReady;

  return api;
}
```

We will assume in the next sections that api is declared and connected.

### Getting the blockhash or block number

For a Block number corresponds a single block hash when the block is finalized, and vice-versa:

```
const blockNumber = "1234567";
const blockHash = await api.rpc.chain.getBlockHash(blockNumber);
const blockHeader = await api.rpc.chain.getHeader(blockHash);

```

```
// Get the latest finalized block hash
// const latestFinalizedHash = await this.api.rpc.chain.getFinalizedHead()

console.log(blockHash); // 0xdb72d44b43f110d4e911f8e2dabc6a8077f984ccaf3ec0e31c934fcf0acce3de
console.log(JSON.stringify(block, null, 2))
```

```
{
  "parentHash": "0xb1e2e4e48c88d13f721329c6447f83de9de26ab959b0d64011b13f0eb3ddc640",
  "number": 1234567,
  "stateRoot": "0xf23a3bb2efd22e703d662e90fa7d3b31ddb0752471e258c8846bcdbf61c8e333",
  "extrinsicsRoot": "0x51de6c0d334c9766d8f76ea310ec0611b756e168862589b1bb5f5e48389339cc",
  "digest": {
    "logs": [
      {
        "PreRuntime": [
          1161969986,
"0x030c00000045c9df0f00000000bee7924ae0598e4de68788793149590bd754e038151545cd5c373e355aacb518e3bf5269ec8d49b48e1a4d6a1193186ee7393d1be616595373d73a93f0b5a50d938013c70a552ddae7ef54af0e989e317fa610d84949638fe75e2f2213e60902"
        ]
      },
      {
        "Seal": [
          1161969986,
          "0x2e5d8b15b4f21eb123a3e27e2f8b2e4db1e24eb4549054d2633399d6dfd5612736d05eec54ba30b7c395beae0d037064f61554c349ec26f6e1117b14fa3e3383"
        ]
      }
    ]
  }
}
```

## Fetching balances of an account

The balances pallets stores balances of all accounts, and provide historic of balances through the blockchain:

```
const ADDR = '12JHbw1vnXxqsD6U5yA3u9Kqvp9A7Zi3qM2rhAreZqP5zUmS';
const balance = await api.query.system.account(ADDR);

// You can fetch data at a specific block hash
// const data = await api.query.system.account.at(blockHash, ADDR);

console.log(JSON.stringify(balance, null, 2));
```

```
{
  "nonce": 34,
  "refcount": 2,
  "data": {
    "free": 85239464459,
    "reserved": 0,
    "miscFrozen": 24000000000,
    "feeFrozen": 24000000000
  }
}
```

You can also use the api-derive methods to fetch multiple balances data at once:

```
const ADDR = '12JHbw1vnXxqsD6U5yA3u9Kqvp9A7Zi3qM2rhAreZqP5zUmS';
const balances = await api.derive.balances.all(ADDR);
```

```
{
  "accountId": "12JHbw1vnXxqsD6U5yA3u9Kqvp9A7Zi3qM2rhAreZqP5zUmS",
  "accountNonce": 34,
  "availableBalance": 61239464459,
  "freeBalance": 85239464459,
  "frozenFee": 24000000000,
  "frozenMisc": 24000000000,
  "isVesting": false,
  "lockedBalance": 24000000000,
  "lockedBreakdown": [
    {
      "id": "0x7374616b696e6720",
      "amount": 24000000000,
      "reasons": "All"
    }
  ],
  "reservedBalance": 0,
  "vestedBalance": 0,
  "vestedClaimable": 0,
  "vestingEndBlock": 0,
  "vestingLocked": 0,
  "vestingPerBlock": 0,
  "vestingTotal": 0,
  "votingBalance": 85239464459
}
```

## Get the staking state of an account

An account can be a stash associated with a controller account and some of its funds can be bonded for staking. A Controller account thus can be associated with a Stash. Some of the transaction must be signed by the Stash (bond for instance), and some must be signed by the Controller (unbond, rebond, nominate, etc...). An account can both be a Stash and Controller.

Thus we need to know those information when interacting with the blockchain.

```
// Obélix accounts topologies
const OBELIX = '12JHbw1vnXxqsD6U5yA3u9Kqvp9A7Zi3qM2rhAreZqP5zUmS'; // OBELIX account 0 - both stash and controller
const OBELIX_STASH = '13b6BF64CN7p42cU4y9N5qWKp6GKGswfzzhA8R3emiNfgAY6'; // OBELIX account 1 is a stash only, controlled by account 2
const OBELIX_CONTROLLER = '13b6BF64CN7p42cU4y9N5qWKp6GKGswfzzhA8R3emiNfgAY6'; // OBELIX account 2 is a controller only for stash account 1

// Might be a controller or a stash or both
const ADDR = OBELIX; // CHANGE THIS TO TEST DIFFERENT SCENARIOS

const controllerOpt = await api.query.staking.bonded(ADDR);
if (controllerOpt.isNone) {
```

```
  console.log(`Address ${ADDR} is not a stash`);
} else {
  console.log(`Address ${ADDR} is a stash`);
}
const CONTROLLER_ADDR = controllerOpt.isSome ? controllerOpt.unwrap() : ADDR;

// Get the staking ledger of a stash - must be called with the controller address
const ledgerOpt = await api.query.staking.ledger(CONTROLLER_ADDR);
if (ledgerOpt.isNone) {
  console.log(`Address ${CONTROLLER_ADDR} is not a controller`);
} else {
  console.log(`Address ${CONTROLLER_ADDR} is a controller`);
}
const ledger = ledgerOpt.isSome ? ledgerOpt.unwrap() : null;

console.log('ledger: ', JSON.stringify(ledger, null, 2));

const STASH_ADDR = ledger ? ledger.stash : ADDR;
console.log('stash address: ', STASH_ADDR);

// Get the nominations for an account at a block hash
const nominations = await api.query.staking.nominators(STASH_ADDR); // use nominators.at(blockHash, ADDR) for a specific block
console.log('nominations: ', JSON.stringify(nominations, null, 2));
```

```
Address 12JHbw1vnXxqsD6U5yA3u9Kqvp9A7Zi3qM2rhAreZqP5zUmS is a stash
Address 12JHbw1vnXxqsD6U5yA3u9Kqvp9A7Zi3qM2rhAreZqP5zUmS is a controller
ledger:  {
  "stash": "12JHbw1vnXxqsD6U5yA3u9Kqvp9A7Zi3qM2rhAreZqP5zUmS",
  "total": 24000000000,
  "active": 14000000000,
  "unlocking": [
    {
      "value": 5000000000,
      "era": 134
    },
    {
      "value": 1000000000,
      "era": 137
    },
    {
      "value": 4000000000,
      "era": 137
    }
  ],
  "claimedRewards": [
    15,
    ...
  ]
}
stash address:  12JHbw1vnXxqsD6U5yA3u9Kqvp9A7Zi3qM2rhAreZqP5zUmS
nominations:  {
  "targets": [
    "14Y4s6V1PWrwBLvxW47gcYgZCGTYekmmzvFsK1kiqNH2d84t",
    "16Dgcx1qJzp8kme1CWsySDf3JWd1oKtbChYVm2yAYqM44woY",
    "12wrtpTVSRqNjC1iyyNjQCEarLH3Lx38S8nKVodRGiHK9CEP",
    "14yx4vPAACZRhoDQm1dyvXD3QdRQyCRRCe5tj1zPomhhS29a",
    "1VrKDfXunzstY5uxPpjArUbZekirGXcpMDYvCBJmjV1KdEm",
    "12Yz9HPcF66pAGpwEW5cyFZ59TFeXFGVnkuxTphC3Lrap29z",
    "12e1tkDgfF3GYdiTkRq1vunXrvvhpKq3BQZYbJ1haXHApQTn"
  ],
  "submittedIn": 119,
  "suppressed": false
}
```

## Get staking progress and session informations

Staking is a pallets in Polkadot dealing with validators selection (during session), nominations, and bonding/unbonding of funds to those validators.

It's important to fetch some of the information about the progress of the sessions to understand how to interact with the blockchain:

```
const blockHash = await api.rpc.chain.getBlockHash(1231000);
const validatorsCount = await api.query.staking.validatorCount.at(blockHash);
const activeEra = await api.query.staking.activeEra.at(blockHash); // must be unwrap()
const currentEra = await api.query.staking.currentEra.at(blockHash); // must be unwrap()
const electionStatus = await api.query.staking.eraElectionStatus.at(blockHash); // must be unwrap()
const validatorSet = await api.query.session.validators.at(blockHash);
```

```
validatorsCount: 197
activeEra = {"index":79,"start":1597851378006}
currentEra = 80
electionStatus.isClose === true
validatorSet = [1Y6WgLRtW6JxmZjSNYsJ9b6JzuF8Kdd7t9kUNiYk9SJXraW, ...196 other values]
```

## Get validators

Validators can be either elected or waiting (not elected) during each era.

Users must pick from any validators to set their nominations list, and will need to have some informations about those validators to choose wisely who to trust in a NPoS consensus blockchain.

```
const activeEraOpt = await api.query.staking.activeEra();
const activeEraIndex = activeEraOpt.unwrap().index;

const allValidatorsAddresses = await api.derive.staking.stashes();
// Under the hood, this is equivalent to getting the keys of all validators:
// const allValidatorsAddresses = await api.query.staking.validators.keys();

// Before an era begins, election might have already run and defined future validators:
const nextElected = await api.derive.staking.nextElected();
// under the hood this is equivalent to:
```

```
// const nextElected = api.query.staking.erasStakers.keys(currentEra)); // with currentEra not always being the active era

// List of elected validators (active era/session)
const electedValidators = await api.query.session.validators();
const validatorAddress = electedValidators[0];
console.log('validatorAddress: ', validatorAddress);

// Get the stake of a validator
const exposure = await api.query.staking.erasStakers(activeEraIndex, validatorAddress);
console.log('exposure: ', JSON.stringify(exposure, null, 2));

// Get the validator preferences (commission)
const prefs = await api.query.staking.validators(validatorAddress);
console.log('preferences: ', prefs);
```

```
validatorAddress:  1FCu68ZwBHNzZLcGa92eHwvR61hk3MpjrhiqN96xF9vWS1Q
exposure:  {
 "total": "0x00000000000000000008462ec88fd526c",
 "own": 99360406100,
 "others": [
  {
   "who": "13HPvN3L6GegLMuPnhgwwQQgp1WKRf9TUxzqWEkZaFLbe1Sa",
   "value": 2644774527851
  },
  {
   "who": "15j4dg5GzsL1bw2U2AWgeyAk6QTxq43V7ZPbXdAmbVLjvDCK",
   "value": "0x0000000000000000000084606d9dcd34ad"
  }
 ]
}
preferences:  {commission: 1000000000}
```

Use api-derive to fetch all information in a single call (can be greedy):

```
const stashIds = await api.query.session.validators();
const validators = await api.derive.staking.accounts(stashIds);

// Or a single value
// const validator = api.derive.staking.account(stashIds[0]);

console.log(JSON.stringify(validators[0], null, 2));
```


```
{
 "nextSessionIds": [
  "128D6Bd4ABX3JHECT1FSuT3TwMS44ntceEL9XzsWyNHrmKkA",
  "16mCVQiVKhYoZvSb49yuJ5nSRZg5XzQCaWBTTBnRvFU8x8GE",
  "1pZs73ci35UBcETVw8r3oBk6q5y7mimuUQdryKteWZ6G6Ap",
  "16JBh5tR35V7CgmYnQ2A9Wnuo8V4G3dWYKhwXp8psP3UqSQH",
  "17Mpmt53d6ZWE8brk23x1F8uxL41psuhsL3HegssKxqz1rq"
 ],
 "sessionIds": [
  "128D6Bd4ABX3JHECT1FSuT3TwMS44ntceEL9XzsWyNHrmKkA",
  "16mCVQiVKhYoZvSb49yuJ5nSRZg5XzQCaWBTTBnRvFU8x8GE",
  "1pZs73ci35UBcETVw8r3oBk6q5y7mimuUQdryKteWZ6G6Ap",
  "16JBh5tR35V7CgmYnQ2A9Wnuo8V4G3dWYKhwXp8psP3UqSQH",
  "17Mpmt53d6ZWE8brk23x1F8uxL41psuhsL3HegssKxqz1rq"
 ],
 "accountId": "1FCu68ZwBHNzZLcGa92eHwvR61hk3MpjrhiqN96xF9vWS1Q",
 "controllerId": "13gAjcC56upxCdaNS69jB7rvGDzimcztTm9Jxhfgma8w9jor",
 "exposure": {
  "total": "0x00000000000000000008462ec88fd526c",
  "own": 99360406100,
  "others": [
   {
    "who": "13HPvN3L6GegLMuPnhgwwQQgp1WKRf9TUxzqWEkZaFLbe1Sa",
    "value": 2644774527851
   },
   {
    "who": "15j4dg5GzsL1bw2U2AWgeyAk6QTxq43V7ZPbXdAmbVLjvDCK",
    "value": "0x0000000000000000000084606d9dcd34ad"
   }
  ]
 },
 "nominators": [],
 "rewardDestination": {
  "Stash": null
 },
 "stakingLedger": {
  "stash": "1FCu68ZwBHNzZLcGa92eHwvR61hk3MpjrhiqN96xF9vWS1Q",
  "total": 99360406100,
  "active": 99360406100,
  "unlocking": [],
  "claimedRewards": [
   38,
   ...
  ]
 },
 "stashId": "1FCu68ZwBHNzZLcGa92eHwvR61hk3MpjrhiqN96xF9vWS1Q",
 "validatorPrefs": {
  "commission": 1000000000
 },
 "redeemable": 0
}
```


## Get the identity of an account

Identity is a pallet in Polkadot providing additional information about an account like its name, and contact infos (email, website, twitter, etc...).
It's a flexible data structure encoded as Raw and there is no specification on the fields name and format.
An identity can also be linked to a parent identity (superOf) and judged by a Registry as valid (*Reasonnable*) or invalid.
The easiest way to fetch identity is to use `api-derive` as it defines common usage of the identity pallet:

```
const ADDR = "16Dgcx1qJzp8kme1CWsySDf3JWd1oKtbChYVm2yAYqM44woY";

// Quick raw data to string - not to be used
function u8aToString(uint8Arr) {
  return String.fromCharCode.apply(null, uint8Arr);
}

// copy/paste from api-derive sources
function dataAsString(data) {
  return data.isRaw
    ? u8aToString(data.asRaw.toU8a(true))
    : data.isNone
    ? undefined
    : data.toHex();
}

const superOfOpt = await api.query.identity.superOf(ADDR);
if (superOfOpt.isSome) {
  console.log(`${ADDR} has a parent identity`);
  const superOf = superOfOpt.unwrap();
  console.log('display: ', dataAsString(superOf[1]));
  const parentIdentity = await api.query.identity.identityOf(superOf[0]);
  console.log('parent address: ', superOf[0]);
  console.log('parent display: ', dataAsString(parentIdentity.unwrap().info.display));
  console.log('parent raw identity: ', JSON.stringify(parentIdentity, null, 2));
} else {
  const rawIdentity = await api.query.identity.identityOf(ADDR);
  console.log('display: ', dataAsString(rawIdentity.unwrap().info.display));
  console.log('raw identity: ', JSON.stringify(rawIdentity, null, 2));
}

// Easiest way to retrieve identity
const { identity } = await api.derive.accounts.info(ADDR);
console.log('decoded identity: ', JSON.stringify(identity, null, 2));
16Dgcx1qJzp8kme1CWsySDf3JWd1oKtbChYVm2yAYqM44woY has a parent identity
display:  Bison Trails 2
parent address:  12EvNGseaJLHmhSKnGdqsG8DC9LNpGUYXwBeU7jUengEG9EN
parent display:  Bison Trails
parent raw identity:  {
 "judgements": [],
 "deposit": 202580000000,
 "info": {
   "additional": [],
   "display": {
     "Raw": "0x4269736f6e20547261696c73"
   },
   "legal": {
     "Raw": "0x4269736f6e20547261696c73"
   },
   "web": {
     "Raw": "0x68747470733a2f2f6269736f6e747261696c732e636f"
   },
   "riot": {
     "Raw": "0x406269736f6e747261696c733a6d61747269782e6f7267"
   },
   "email": {
     "Raw": "0x737570706f7274406269736f6e747261696c732e636f"
   },
   "pgpFingerprint": null,
   "image": {
     "None": null
   },
   "twitter": {
     "None": null
   }
 }
}
decoded identity:  {
 "display": "Bison Trails 2",
 "displayParent": "Bison Trails",
 "email": "support@bisontrails.co",
 "judgements": [],
 "legal": "Bison Trails",
 "other": {},
 "parent": "12EvNGseaJLHmhSKnGdqsG8DC9LNpGUYXwBeU7jUengEG9EN",
 "riot": "@bisontrails:matrix.org",
 "web": "https://bisontrails.co"
}
```