GETTING STARTED WITH BLOCKCHAIN DEVELOPMENT

So you want to build a blockchain application? Welcome to the wonderfully complex and incredibly powerful world of crypto-currencies and distributed digital assets. Hold on tight.

First things first, if you do not already have some coins to play with, we'll need to get some.

We recommend you start with the Dogecoin Testnet as it's incredibly easy to get hold of the necessary coins and for you to sync its' blockchain. They also work in much the same way as Bitcoin, so anything you can learn from a Bitcoin tutorial will usually apply to Dogecoin too.

1) - Download DogecoinQT (must choose full-client) from here - http://dogecoin.com

2) - Install, open and then close the DogecoinQT (we need to alter the config to switch to test-mode but cannot do that until we have first launched the program)

3) - Create a dogecoin.conf file in the relevant folder created when installing the QT

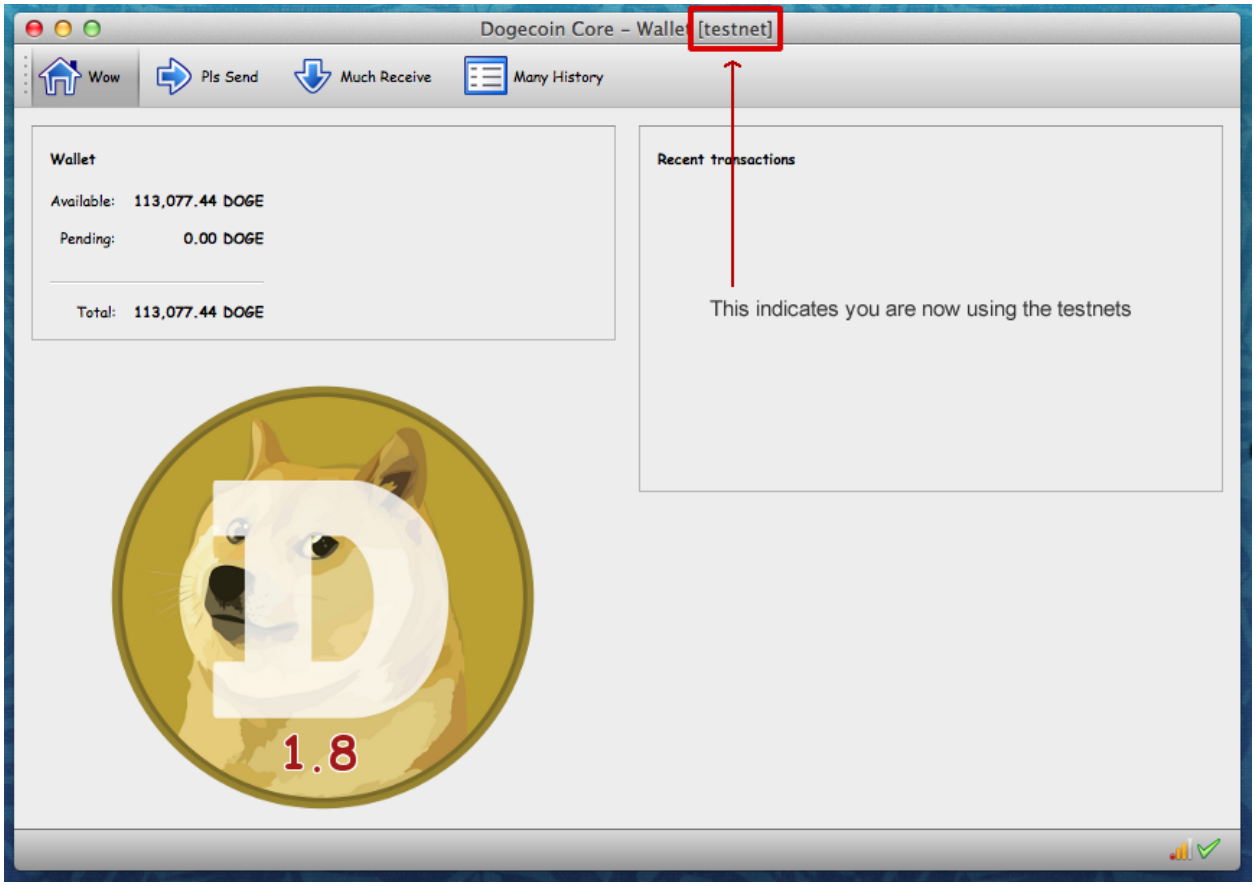
4) - Add the following information into the dogecoin.conf file:

```
txindex=1
testnet=1
gen=0
```

The **txindex** setting allows your node to index all transactions, rather than just your own, whereas the **testnet** switches the QT from the main (production) network to the test-mode. If you want to start mining coins (best to do overnight as it is resource heavy) simply change the **gen=0** setting to **gen=1** and when you are finished testing and want to use real coins, again, just change the **testnet=1** setting to **testnet=0**. The configuration files for Blockchain 1.0 technologies use the same settings. If you want to be able to access the QT remotely from your server-side language of choice, add these:

```
server=1
rpcport=44555
rpcuser=admin
rpcpassword=ChooseYourOwnButItMustBeReallyLong
```

You can choose whatever port numbers, username or passwords you want here, but for now, the important part is that we are sure we are using the testnet mode. Once you are happy with the configuration, reopen the DogecoinQT and you should see the following:



4) - Once you are sure you are using the correct network, you'll next need to create an address by clicking on "**File > Much receiving addresses**".

From here, you can create as many new addresses as you like and label them as you do. When you have an address from which you can receive coins, you can then try to use the various online Faucets designed for that network. Faucets supply FREE coins but often involve clicking on various advertisements and other obtrusive obstacles.

5) - We have our own multi-currency Faucets you can try - http://faucets.blockstrap.com - all you need to do is enter an email address to where we can send you a claim code.

So now that you have some coins, it's time to get started with our stack. We provide a set of complete turn-key solutions, from front-end frameworks to back-end APIs and SDKs.

All of our products and services support the following blockchains:

Blockchain	**Website**	**Main Code**	**Test Code**
Bitcoin	http://bitcoin.org	btc	btct
DashPay	http://dashpay.io	dash	dasht
Dogecoin	http://dogecoin.com	doge	doget
Litecoin	http://litecoin.org	ltc	ltct

Our API is the most important component of the our stack, without it, you will need to manage your own blockchain nodes and probably even parse them into a database for easier fetching.

To check the details of a Bitcoin address (in this example, the address we are checking is **1JsoyFgFugGRRY7qkPGTHaKVQpeqf67VVb**) using our API, simply visit the following URL:

http://api.blockstrap.com/v0/btc/address/transactions/1JsoyFgFugGRRY7qkPGTHaKVQpeqf67VVb

The JSON results returned should contain information related to this address. In order to see additional transaction details, you will need to add the **?showtxnio=1** parameter to the URL.

Information on this & other web-accessible functions can be seen in our API documentation:

http://docs.blockstrap.com/en/api/

For simplified access to our API, we provide server-side SDKs in the following languages:

- PHP - http://github.com/blockstrap/blockstrap-php
- Ruby - http://github.com/blockstrap/blockstrap-ruby

If you are looking for a little more help, you may want to consider taking a look at one of our sample applications, which are also open-source and provide a solid foundation to start from.

We currently provide support for the following applications:

- **Deterministic Web-Wallet** (HTML):
  Demo: http://demo.blockstrap.com | Source: http://github.com/blockstrap/framework

- **Multi-Currency Block-Explorer** (PHP):
  Demo: http://blockchains.io | Source: http://github.com/blockstrap/blockchains.io

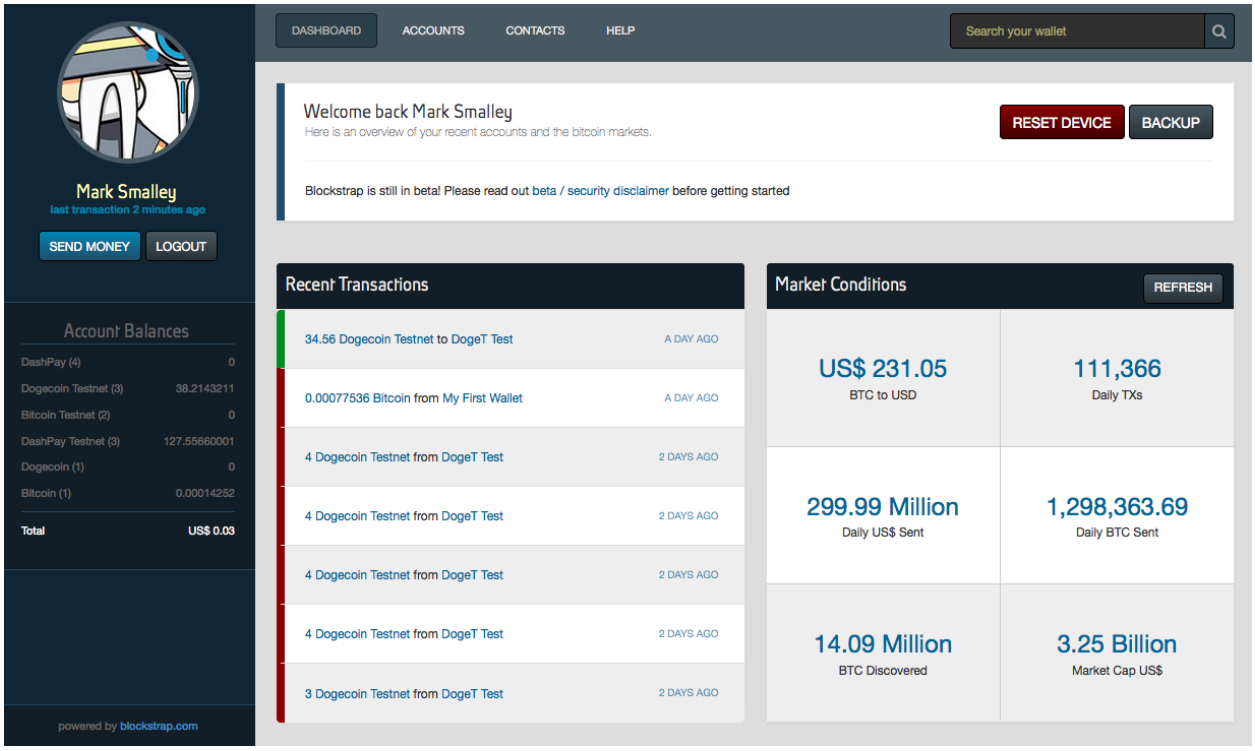
- **Blockchain File Uploader** (PHP):
  Demo: http://uploads.blockstrap.com | Source: http://github.com/blockstrap/uploads

- **BlockAuth: Self-Managed Registration & Authentication** (PHP):
  Demo: http://blockauth.org | Source: http://github.com/Neuroware-IO/blockauth

- **Email-Driven (Lead-Capturing) Multi-Currency Faucets** (PHP):
  Demo: http://faucets.blockstrap.com | Source: http://github.com/blockstrap/faucets

Please note that all of our sample applications also use our JS Framework, which can not only be used as a standalone platform for browser-based blockchain development, but can also be included with server-side projects and used to simplify the creation of transactions, as seen below in this example that will send yourself a transaction whilst encoding additional data into the blockchain using the OP_RETURN method.

By default, downloading and running our Framework - http://github.com/blockstrap/framework - frm a server, will result in the following default wallet theme being activated:



It too supports all eight blockchains, features simplified account and contact management, without storing or transmitting any private information anywhere.

Nonetheless, it can also be configured in such a way that it is used to complement other server-side projects as mentioned above, and will now demonstrate some sample code:

```
var keys = $.fn.blockstrap.blockchains.keys('SECRET_SALTED_HASH','btc');
$.fn.blockstrap.api.unspents(keys.pub,'btc', function(unspents)
{
	if($.isArray(unspents) && blockstrap_functions.array_length(unspents) > 0)
	{
		var total = 0; var inputs = [];
		var fee = $.fn.blockstrap.settings.blockchains['btc'].fee;
		$.each(unspents, function(k, unspent)
		{
			inputs.push({
				txid: unspent.txid,
				n: unspent.index,
				script: unspent.script,
				value: unspent.value,
			});
			total = total + unspent.value
		});

		var outputs = [{
			address: keys.pub,
			value: total - fee
		}];

		var raw_tx = $.fn.blockstrap.blockchains.raw(
			keys.pub,
			keys.priv,
			inputs,
			outputs,
			fee,
			total - fee,
			'Testing Data Encoding'
		);
		$.fn.blockstrap.api.relay(raw_tx,'btc', function(results)
		{
			// Your callback, results should be a TXID (if successful)
			console.log(results);
		});
	}
});
```