Borrow That (Mango release, 13/05/16)

By CHAKAS (Ario Aliabadi, Cataldo Azzariti, Lottie Carruthers, Joshua Howarth, Veronika Kolejakova, Kayla Shapiro)

Borrow That is a proof-of-concept Rails web application that utilizes *Distributed Ledger Technology (DLT)* to implement a decentralised peer-to-peer sharing model. Through the app, users can lend or borrow items from each other. The transactions are facilitated by the use of *'smart contracts'* which encapsulate the agreement between the lending and borrowing users. The process of writing, signing and verifying a contract is completely automated. Each transaction is registered into a block and added onto a ledger. This ledger is implemented through the use of an *Ethereum blockchain*. Once the code of the smart contract is on the blockchain, it can be executed, but not altered. In this way, we provide a platform that is secure, transparent, and convenient.

Installation

App Dependencies

To run the app, the first thing you will need to do is install RVM (Ruby Version Manager).

While it is best practice to research the current best installation practices, these are the terminal commands that worked best for us:

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3
```

```
$ \curl -sSL https://get.rvm.io | bash -s stable
```

At this point, it may be necessary to run `rvm reload` to reload RVM source and restart the virtual machine.

To run a Ruby on Rails based application, we really need to have Ruby, Rails, and some sort of database management system (in our case, we chose Postgres).

Install ruby version 2.3.0:

- `rvm install 2.3.0`
- rvm --default use 2.3.0

Install PostgreSQL:

- `gem install pg`

Install Rails:

(WARNING: this is a long install; to see installation steps, add " `-v`" to the end of the following command)

- `gem install rails`

The app is located in the directory "`p2pv2`". To run the application, one needs to enter this directory.

1. Run "`bundle install`" to install libraries and their dependencies
2. Run "`rake db:create`" to create your own development database
3. Run "`rake db:migrate`" to produce a correct database schema

This concludes installation for the Rails App. For the purposes of our proof-of-concept we do not download the entire blockchain (which is huge). Instead, we can run a "testchain."

Cloud 9 IDE Setup PSQL Database Setup

In order to develop in the Cloud 9 IDE (which can aid in rapid development and prototyping particularly with its inbuilt console, file tabs and file management system), one must install and setup PSQL in their environment and create a username and password.

1. Create a new username and password for postgresql on cloud9: `$ sudo service postgresql start $ sudo sudo -u postgres psql postgres=# CREATE USER username SUPERUSER PASSWORD 'password'; postgres=# \q`
2. Create ENV variables on cloud9: `$ echo "export USERNAME=username" >> ~/.profile $ echo "export PASSWORD=password" >> ~/.profile $ source ~/.profile`
3. My database.yml for rails 4.2.0 on cloud9: `default: &default adapter: postgresql encoding: unicode pool: 5 username: <%= ENV['USERNAME'] %> password: <%= ENV['PASSWORD'] %> host: <%= ENV['IP'] %> development: <<: *default database: sample_app_development test: <<: *default database: sample_app_test production: <<: *default database: sample_app_production`
4. Include the gem pg in Gemfile and install: gem 'pg', '~> 0.18.2' and `$ bundle install`
5. Update template1 postgresql for database.yml on cloud9: `postgres=# UPDATE pg_database SET datistemplate = FALSE WHERE datname = 'template1'; postgres=# DROP DATABASE template1; postgres=# CREATE DATABASE template1 WITH TEMPLATE = template0 ENCODING = 'UNICODE'; postgres=# UPDATE pg_database SET datistemplate = TRUE WHERE datname = 'template1'; postgres=# \c template1 postgres=# VACUUM FREEZE; postgres=# \q`
6. From command line run: `bundle exec rake db:create`

Testchain Dependencies

We begin by installing `geth` which is a command line interface for running a full Ethereum node implemented in Go. An Ethereum node is a piece of software that stores the blockchain and connects to other nodes. These nodes form the Ethereum network.

With geth, one can mine ether, create contracts to send transactions and explore block history. We used the following command to install geth:

```
bash <(curl -L https://install-geth.ethereum.org)
```

Next, to store all of the information about the blockchain, create a directory. Our directory was called "`ethereum_test`". We will have to store two things inside this directory. The first is a genesis block. This is basically a database file. When a user puts it into the client, it represents their decision to be a part of the Ethereum network. A script exists to generate this block. But, to simplify the process, the following should be stored in a file called `genesis.json`:

```
{
        "nonce": "0xdeadbeefdeadbeef",
        "timestamp": "0x0",
        "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
        "extraData": "0x0",
        "gasLimit": "0x8000000",
        "difficulty": "0x400",
        "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
        "coinbase": "0x3333333333333333333333333333333333333333",
        "alloc": {
        }
}
```

The second thing needed in the directory, is a further directory to store all of the data for the testchain. We named our directory "`data`".

Now we are ready to start our Ethereum node. To launch geth with the testchain:

```
geth --networkid 9999 --genesis PATH/genesis.json --rpc --rpcport "9500" --rpcaddr
"0.0.0.0" --rpccorsdomain "*" --datadir PATH/data console
```

The RPC port is specified in order to make calls from the web application to the Ethereum node and vice versa.

Borrow That assumes that users are already connected to their Ethereum account. In order to create an account in geth, enter the console by typing "`geth`". Once inside, one can create an account by entering "`eth.newAccount()`". The user will be prompted to enter a new password for their account. This new account will contain no ether. To get ether, one can mine the blockchain with "`miner.start()`". Be aware that it can take up to an hour for the chain to start mining. Once a few blocks have been mined, the user can stop the miner with "`miner.stop()`".

Usage

One of our gems, browserify, adds Javascript module support to sprockets (the gem used in the Rails asset pipeline to compile and serve web assets). This allows our application to make use of the web3 Javascript API. To get browserify to work with the assets, one must run: "`npm install`".

Now that all of the necessities have been installed, it is time to get the application up and running. To make use of the Rails asset pipeline, we will precompile all of the Javascript files. To do this, run: “`rake assets:precompile`”. This speeds up the loading of the web application.

To get the web application up and running, enter “`rails server`” into the terminal. The app will be available in development mode at localhost:3000.

In order to execute contracts, the testchain must be mining.