vue-dark-chocolate¶

Atruffle box that comes with everything you need to start using smart contracts from aVue App withBootstrap styling and components. Supporting frameworks includeVuex ,Vue-Router & Bootstrap 4 (via Bootstrap-Vue) andwebpack . The sample useswebpack-dev-server for hot-reloading or you can build and serve fromexpress .

This truffle box has a simplistic smart contract model for registering/tracking trades between two accounts, written in Solidity. A vendor account registers the trade and the counterparty account has to certify it. The logic is implemented in the smart contracts to support the full trade flow but the frontend is not fully wired-up. A good exercise is to finish wiring up the frontend to go through the entire flow of registering a trade and having the counterparty certify. This provides a complete view of reading and writing to smart contracts from a dApp via truffle and web3 APIs. The sample also uses a few web3 utility methods that demonstrate how you can convert between strings/hex values from the frontend JavaScript all the way through to Solidity smart contracts on the Ethereum blockchain.

This has only been tested on Windows butshould work just as well on Linux(said no one ever) . Restart-TestRpcInstance.ps1 is the only real Windows specific item, unless you havePowerShell on Linux . That said, it just stops/starts a testrpc instance with defaults geared towards easily running this box so should be fairly easy to adapt.

To help reduce the learning curve to get started with Ethereum Blockchain and Truffle suite this sample includes some test data (in the form of testrpc accounts) and data seeding for smart contracts. To get started, you'll need to load the accounts in to Metamask. Check out testrpc-sample.md and use Metamask to import the HD Wallet mnemonic.

Usage¶

If you landed here you probably already have truffle, but just in case:npm install -g truffle

To initialize a project with this example, runtruffle unbox kierenh/vue-dark-chocolate inside an empty directory.

The Ethereum test servernpm install -g ethereumjs-testrpc

Installmetamask (see also:Truffle and Metamask)

Building and running smart contract unit tests¶

1. Start an rpc instanceRestart-TestRpcInstance.ps1 -networkID: 27666
2. (27666 is the ID of the network which is used at unit test time to skip data seeding, see truffle.js config)
3. Runtruffle compile
4. Then runtruffle test --network development-no-data-seed
5. to deploy and test the contracts on the 'development-no-data-seed' network

Highly recommend checking out thetesting doc. At the very least, it's good to be aware of the CLEAN-ROOM ENVIRONMENT approach:

CLEAN-ROOM ENVIRONMENT Truffle provides a clean room environment when running your test files. When running your tests against the TestRPC, Truffle will use the TestRPC's advanced snapshotting features to ensure your test files don't share state with each other. When running against other Ethereum clients like go-ethereum, Truffle will re-deploy all of your migrations at the beginning of every test file to ensure you have a fresh set of contracts to test against.

Building and running (interact via truffle console)¶

1. Start an rpc instanceRestart-TestRpcInstance.ps1
2. Then runtruffle compile
3. , then runtruffle migrate
4. to deploy the contracts onto your network of choice (default "development").
5. Then runtruffle console
6. Seesample_console.md
7. which shows how you can interact with the contracts via the console
8. To exit.exit

Building and running the frontend (hot-reloading)¶

1. Start an rpc instanceRestart-TestRpcInstance.ps1
2. Then runtruffle compile
3. , then runtruffle migrate
4. to deploy the contracts onto your network of choice (default "development").
5. Then runnpm run dev

6. to build the app and serve it on http://localhost:8081 (with support for hot reloading)

Building and running the frontend (express)¶

1. Start an rpc instanceRestart-TestRpcInstance.ps1
2. Then runtruffle compile
3. , then runtruffle migrate
4. to deploy the contracts onto your network of choice (default "development").
5. Then runnpm run build
6. (see the output in the build folder)
7. Then runnpm start
8. to start the express server (serves the static content from the build folder)

On migrations¶

In combination with smart contract design, how migrations work, and how you reference deployed contracts is fairly fundamental, so just wanted to callout these docs to accelerate your learning (hopefully).

The Migrations contract stores (in last_completed_migration) a number that corresponds to the last applied "migration" script, found in the migrations folder. Deploying this Migrations contract is always the first such step anyway. The numbering convention is x_script_name.js, with x starting at 1. Your real-meat contracts would typically come in scripts starting at 2_....

So, as this Migrations contract stores the number of the last deployment script applied, Truffle will not run those scripts again. On the other hand, in the future, your app may need to have a modified, or new, contract deployed. For that to happen, you create a new script with an increased number that describes the steps that need to take place. Then, again, after they have run once, they will not run again. what are truffle migrations on stackoverflow

That's right, they won't run again, ever, so make sure your contract is ready for the wild because once it's on the blockchain, it's there forever! You can't really pull it back and add a field like you might with a database table :)

When compiling and running migrations on a specific network, contract artifacts will be saved and recorded for later use. When your contract abstractions detect that you're Ethereum client is connected to a specific network, they'll use the contract artifacts associated that network to simplify app deployment. truffle docs - advanced - configuration#networks

Possible upgrades¶

Upgrades welcome, just shoot through a pull request.

- Listen to events in the TradeRegister.sol and read historical events via web3 event/filter APIs
- Build and Release from VSTS to an Azure App ServiceETHEREUM DEVOPS WITH TRUFFLE, TESTRPC & VISUAL STUDIO TEAM SERVICES
- &Ethereum DevOps with VSTS – easier now with new Truffle installer + npx

Errors and Troubleshooting¶

Chrome, Metamask & TestRPC¶

After going through the dev-test inner-loop cycle many times, there was an instance where the web app did not retrieve the correct data from the blockchain. There were 2 trades loaded, but I could only retrieve 1, and only for some accounts.

- In the web app, I could see it was not retrieving the correct number of trades for each actor (and subsequently could not look them up * correctly) by debugging the Javascript
- In truffle console, I was able to run ad-hoc queries and get the expected results
- Truffle test - the unit tests were all passing
- I'm fairly certain I closed/re-opened browser during this time, but what seemed to do the trick was forcing metamask to re-connect to the testrpc instance so I have put this down to a quirk in metamask connecting to testrpc (certainly the versions being used anyway).

When I re-selected the network, the page refreshes and the web app returns the correct results. Basically this was just opening metamask and clicking on "Localhost 8545" which seemed to trigger some kind of refresh and things started working as expected. This post reports a similar problem: https://stackoverflow.com/questions/45585735/testrpc-the-tx-doesnt-have-the-correct-nonce There doesn't appear to be a great solution, the advice is to close browser instances etc and reconnect to the RPC.

truffle test¶

It's important to be aware that tests that don't handle exceptions thrown from within promises correctly can cause

subsequent tests to fail with misleading error messages. Perhaps tests in 1 suite work, but when combined with the whole test suite they fail. This is a good indication there's an exception thrown from within a promise is not handled correctly. Be sure to add a catch() handler to the top-level promise. To help troubleshoot, you can isolate tests by running individual files in Truffle.truffle test ./path/to/test/file.js

Dev Notes and Links¶

I'll leave the bulk of this to some of the resources I found helpfulsolidity docs

web3js 1.0 docs

webpack A 101 Noob Intro to Programming Smart Contracts on EthereumBlockchain fundamentals on PluralSight bytes32 VS string types on stackoverflow Solidity - Smart Contract Design - When to use contract and when to use structs on stackoverflow Get started with vue-bootstrap