blockchain Documentation

Release unknown

Sebastian Jäger

Feb 09, 2020

Contents

This is the documentation of **Blockchain** a simple implementation in Python to get familiar with Python and the basic concepts of Blockchains.

Short Disclaimer: It is just a private `Python 3.7.2` project. Its purposes is to get a little bit familiar with the Python projects and the concepts of Blockchains. Therefore it is not intended for production usage, and any warranties are excluded.

CHAPTER 1

Getting Started

The easiest way to get up a single miner or a whole blockchain network is to use Docker. This repository offers the needed `Dockerfile` and `docker-compose.yaml` in the directory `docker`. Do the following steps:

1. Change to `docker` directory
2. Run `docker build --no-cache -t blockchain .`
3. Run `docker-compose up`

This starts a Blockchain network with 3 miners and forwards their ports (12345, 12346, 12347) to your host system. It uses the directory `~/.blockchain/` on your host system to save the created files for each miner.

1.1 Install the CLI Locally

1. Clone this repository: `git clone git@github.com:se-jaeger/blockchain.git`
2. Open the clone directory: `cd blockchain`
3. Create a virtual env: `python -m venv venv`
4. Activate the virtual env: `source venv/bin/activate`
5. Install all dependencies: `pip install -r requirements.txt`
6. Install the `blockchain` CLI, run the following in the root directory of this project: `pip install -e .`
7. Check available commands: `blockchain --help`

CHAPTER 2

How Does This Blockchain Implementation Work?

This implementation produces a simple `CLI`, `Miner` and `UI`. It is necessary to get up and running a local Miner. The CLI, as well as the UI, uses the Miners `REST` interface to interact with it. Created `messages` get synchronized with all other known Miners (`neighbours`) in the Blockchain network. A Miner asks all its neighbours periodically (if not max amount of neighbours is reached) to send unknown Miner and connects to them. Also in a periodical manner, Miner synchronizes their local Blockchain with the chains of there neighbours and use the longest valid chain in the network.

2.1 Miner Implementation

This Miner implementation offers a REST API with the following endpoints:

- **`/add` (PUT): needs the URL parameter `message`. Adds the message to the local cache of unprocessed data.**
  - `response` (200): JSON with message: 'Message added!'
  - `response` (400): JSON with message: 'No Message added!'
- **`/chain` (GET): Returns the miners local chain.**
  - `response` (200): JSON with the actual chain and its length.
- **`/neighbours` (GET): Returns the miners neighbours.**
  - `response` (200): JSON with the actual neighbours and its length.
- **`/data` (GET): Returns the miners local cache of unprocessed data.**
  - `response` (200): JSON with the actual list of unprocessed data.

The miner uses a set of files for normal operation:

- <filename>.chain: Representation of the actual file.
- <filename>.hash: SHA-256 of the actual chain file. Is used to check if the local chain differs from its on disc representation.
- <filename>_<date>_<time>: Older versions of the chain file. Created at <date>_<time>.
- miner.log: Log file and up to three backup files named miner.log.x where x is a number.

The Miner runs several Threads and a Process to run parallel and periodical tasks:

- Gossip Job (Thread): Implementation of a simple Gossip Protocol. Fetches periodical all neighbours of its neighbours.
- Sync Chain Job (Thread): To get the actual longest global chain. Fetches periodical the chain of all neighbours.
- Sync Unprocessed Data Job (Thread): To propagate unprocessed data through the network. Fetches periodical the set of unprocessed data of all neighbours.
- Backup Local Chain Job (Thread): To backup the local chain to disc. Backups periodical the local chain to disc if they differ from each other.
- Server Process (Process): Servers the Miners REST API in a separate process.
- Communication Job (Thread): Communication thread to exchange message with the server process.

2.2 Web-based User Interface

The CLI offers a subcommand ui, this allows to start an webserver for convenient interaction with the blockchain system.

2.3 Proof of Work

A very simple implementation of a Proof of Work algorithm. The SHA-256 hash value of the concatenation of the previous proof and the proof of the new Block has to start with difficulty trailing 0s.

CHAPTER 3

Improvements

- Miner endpoint (health) to check availability and provide opportunity to delete a neighbour
- More Error handling -> chain probably gets corrupt when killing miner
- Use locking for (chain, neighbours, data)

CHAPTER 4

Contents

4.1 License

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying

the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "{}" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright {yyyy} {name of copyright owner}

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

4.2 Contributors

- Sebastian Jäger <se.jaeger@web.de>

4.3 Changelog

4.3.1 Version 0.1

- Implement core functionality

4.3.2 Version 0.2

- Implement Web-based user interface

4.4 src

4.4.1 blockchain package

Subpackages

blockchain.blockchain package

Submodules

blockchain.blockchain.block module

class blockchain.blockchain.block.**Block**(*index: int*, *data: blockchain.blockchain.data.Data*, *proof: int*, *previous_hash: str*)

Bases: object

__bytes__() → bytes

Uses the encoded string representation of this Block object as bytes representation.

Returns byte representation of Block object.

Return type bytes

__eq__(*other: object*) → bool

Method for comparing two Block objects.

Parameters **other** (Block) – Block object to compare with self.

Returns True if blocks are equal. False otherwise.

Return type bool

__repr__() → str
String representation of `Block` object.

Returns String representation of `Block` object.

Return type str

data

index

previous_hash

proof

timestamp

blockchain.blockchain.blockchain module

class blockchain.blockchain.blockchain.**Blockchain**(*path_to_chain: str*, *json_format: bool*, *force_new_chain: bool*)

Bases: `object`

_load_chain() → None
Helper method to load chain from disk. Raises an error if no chain is found.

Raises `ChainNotFoundError` – Will be raised if no local chain could be found.

add_new_block(*data: blockchain.blockchain.data.Data*, *proof: int*, *previous_hash: str*) → blockchain.blockchain.block.Block
Adds a new Block to the existing chain.

Parameters

- **data** (`Data`) – Data that is attached to this block.
- **proof** (*int*) – The `proof` value for this block.
- **previous_hash** (*str*) – Hash value of previous block in chain.

chain

genesis_block = | ==========

genesis_block_hash = 'beac2e974625627e92f58831a56fd005570fb08a740cf114deb358dffa6b9525

json_format

last_block

path_to_chain

save_chain() → None
Helper method to save chain to disk. Creates intermediate directories and backups an existing chain file if necessary.

blockchain.blockchain.data module

class blockchain.blockchain.data.**Data**(*message: str*)

Bases: `object`

__hash__()
Needed to use `Set``s of ``Data` objects.

id

message

Module contents

blockchain.cli package

Submodules

blockchain.cli.cli module

Module contents

blockchain.client package

Submodules

blockchain.client.miner module

class blockchain.client.miner.**Miner**(*path_to_chain: str*, *json_format: bool*, *port: int*, *difficulty: int*, *neighbours: list*, *force_new_chain: bool*)

Bases: object

_backup_local_chain() → None
Periodical thread to backup the local chain to disc.

_check_for_longest_chain() → None
Consensus Algorithm:

Ask each neighbour for that neighbours. Add all unknown miner to neighbours set until maximum amount of neighbours is reached.

_communicate() → None
Periodical thread to communicate with server process.

_fetch_unprocessed_data() → None
Periodical thread to get unprocessed data form neighbours. => Broadcasts unprocessed data around the network.

static _hash(*block: blockchain.blockchain.block.Block*) → str
Hash a Block object with SHA-256.

Parameters **block** (Block) – Object of class Block to hash.

Returns Hex representation of block hash.

Return type str

Raises ValueError – Will be raised if no Block object is passed.

_is_chain_valid(*chain: list = None*) → bool

Checks if the given chain satisfies the following rules:

1. **The first (genesis) block:**
   - index = 0

- `previous_hash` = None
- `proof` = None

2. **each and every following block:**
   - `index`: step size 1 and monotonically increasing (1, 2, 3, 4, . . .)
   - `previous_hash`: SHA-256 of the string representation of the preceding block
   - `proof`: has to be valid -> see: `is_proof_of_work_valid()`
   - `timestamp`: higher than the timestamp of of preceding block

Parameters **chain** (*list*) – Optional chain if `None` internal representation is used.

Returns `True` if `chain` is valid, `False` otherwise.

Return type bool

_is_data_processed(*data: blockchain.blockchain.data.Data*) → bool
Checks if `data` is already in local chain.

Parameters **data** (`Data`) – `Data` object to check if it exists in the actual chain.

Returns `True` if unprocessed.

Return type bool

static _is_proof_of_work_valid(*last_proof: int*, *proof: int*, *difficulty: int*) → bool
Checks if the proof of work was correct. The hash value of `last_proof` concatenated with `proof` has to be `difficulty` trailing 0s.

Parameters

- **last_proof** (*int*) – Value of the `proof` of the preceding block.
- **proof** (*int*) – `proof` of the actual block.
- **difficulty** (*int*) – Amount of trailing 0s.

Returns `True` if proof of work is correct, `False` otherwise.

Return type bool

Raises `ValueError` – Will be raised if `difficulty` is not a positive integer value.

_mine() → None
Blocking Mining loop.

If `not_processed_messages` are available it uses a random message an mines a new block.

_new_message(*message: str*) → None

Adds the new `message` to its local cache.

Parameters **message** (*str*) –

_proof_of_work(*last_proof: int*, *difficulty: int*) → int
Simple proof of work:

Find a number `p` that when hashed with the previous `block`'s solution a hash with `difficulty` trailing 0s is produced.

Parameters

- **last_proof** (*int*) – Solution of the last blocks' proof of work
- **difficulty** (*int*) – Amount of trailing 0s for a valid proof of work.

Returns Solution for this proof of work quiz.

Return type int

Raises `ValueError` – Will be raised if `difficulty` is not a positive integer value.

_update_neighbours() → None
Periodical thread to update neighbours if limit is not exceeded.

blockchain

difficulty

jobs

neighbours

port

queue

server_process

start() → None
Starts some background `Job` s for the Gossip Protocol, Chain syncing, Data syncing, communication thread as well as the server functionalities as process. Starts the blocking function `mine()`.

stop() → None
Function that gets called when Python was killed. Takes care to shutting down all threads/process and saves the chain to disc.

unprocessed_data

blockchain.client.server module

blockchain.client.server.**start_server**(*queue: multiprocessing.context.BaseContext.Queue*, *port: int*)

Module contents

blockchain.ui package

Submodules

blockchain.ui.forms module

blockchain.ui.routes module

Module contents

blockchain.utils package

Submodules

blockchain.utils.constants module

blockchain.utils.errors module

exception blockchain.utils.errors.**ChainNotFoundError**
Bases: Exception

Error if no local chain could be found.

exception blockchain.utils.errors.**ChainNotValidError**
Bases: Exception

Error if loaded chain is not valid.

exception blockchain.utils.errors.**PortValueError**
Bases: ValueError

Error if given port is out af valid range (1 - 65535).

exception blockchain.utils.errors.**ProgramKilledError**
Bases: Exception

Error if process get killed.

blockchain.utils.utils module

class blockchain.utils.utils.**Job**(*interval: datetime.timedelta*, *execute*, **args*, ***kwargs*)
Bases: threading.Thread

run() → None
Runs the background Job

stop() → None
Stops the background Job.

blockchain.utils.utils.**colorize**(*text: str*, *color: str*) → str

blockchain.utils.utils.**create_proper_url_string**(*host_port: (<class 'str'>, <class 'int'>)*, *path: str*) → str
Takes the internal representation of neighbours and a endpoint path to create a proper URL string for requests.

Parameters

- **host_port** (*str, int*) – Internal representation of IP address/hostname and port combination.
- **path** (*str*) – The endpoint of the API.

Returns Correct URL string for address and path.

Return type str

blockchain.utils.utils.**encode_file_path_properly**(*file_path: str*) → str
Encode each and every input filepath as absolute pathes.

Parameters **file_path** (*str*) – Path to encode properly

Returns Absolute and properly encoded file_path

Return type str

blockchain.utils.utils.**signal_handler**(*signum*, *frame*)
Signal handler used to raise special ProgramKilledError.

Raises ProgramKilledError – To intercept for graceful shutdown.

blockchain.utils.utils.**split_url_string**(*host_port: str) -> (<class 'str'>, <class 'int'>*)
Parses the given URL string and returns the IP address/hostname and the port/default port.

Parameters **host_port** (*str*) – Representation of the miner as URL string, e.g.: 127.0.0.1:12345, miner1:8888, miner, http://localhost, ...

Returns Tuple of IPv4 Address or hostname string and port number.

Return type (str, int)

Raises

- PortValueError – Will be raised if given port is out of range.
- AddressValueError – Will be raised if given address is not a valid IPv4 address or "localhost".

Module contents

Module contents

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

Symbols

A

B

C

D