## Chaincode

Location in project: test/fixtures/chaincode/src/github.com/example\_cc\_2/example\_cc\_2.go

Used to store and retrieve from blockhain information about transactions in form of current document state. Documents are encoded into format of list of five values for every key:value in document [<name-of-dapp>, <name-of-collection>, <id-of-document>, <key>, <value>]. Only allowed type of value is string.

Functions:

func (t \*SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response - A stub for the init method which always returns Success since we dont need any specific action to initialize the ledger

func (t \*SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response - An entry point for the chaincode. Call update, get, or set functions depending on the function name in the chaincode args.

func update(stub shim.ChaincodeStubInterface, args []string) (string, error) - Update stores the new value for the composite key only if the old value for this key from the prestate list is equal to the last value for this key from the ledger. Or if the old value for this key from the prestate list is emty %). returns error in other cases.

Args is a string array where elements should be parts of the solid json string which in order should be unmarshallable into UpdateAsset structure

func set(stub shim.ChaincodeStubInterface, args []string) (string, error) - Set stores the given pair of the composite key and the value on the ledger. If the key exists, it will override the value with the new one. Args is a string array where the first four elements are the parts of the composite key and the fifth one the value

func get(stub shim.ChaincodeStubInterface, args []string) (string, error) - Get returns the value of the specified composite key. Args is a string array where the first four elements are the parts of the composite key and the fifth one the value

## Polling

Location in project: polling.py

Every 0.1 seconds asks blockhain for a new blocks. In case there are new blocks – extracts transactions from them, decodes key-value as a list of 5 strings back to documents and makes transactions to MongoDB. To memorize which blocks are already decoded and pushed in db it stores information about number of processed blocks into db itself.

## Wallet dApp

Location in project: wallet-dapp.py

Allows creation, getting information and transfering of money between accounts

Methods:

@app.route('/account/<user\_id>')

def account(user\_id): - gets information about user balance in next format: {“key”: <user\_id>, “value”: <balance>}

@app.route('/transfer', methods=['POST'])

def transfer(): - transfers balance by user request in next format: {“from\_user\_id”: <user\_id>, “to\_user\_id”: <user\_id>, “amount”: <amount>}. Response in next format: {'from\_user\_id': <user\_id>, 'balance': <remaining\_balance>}

@app.route('/create\_wallet', methods=['POST'])

def create\_wallet(): - creates account balance by user request in next format: {“user\_id”: <user\_id>, “amount”: <amount>}.

## Custom api for MongoDB

Location in project: color\_mongo.py

Custom mongo api allows us to abort transactions before commiting in MongoDB and instead encoding transactions and sending them into blockchain. All operations taken from standart api are working like proxy to mongo (plus our own logic) except commit of transaction.

Methods:

def insert\_one(self, document): - appends document to list of inserts

def update\_one(self, filter, update): - appends update operation to list of updates. Saves info about prestate

def find\_one(self, filter): - saves info about prestate

def start\_transactions(self): - only proxy to mongodb

def \_convert\_data(self): - Our custom method. converts data from inserts and updates into desired format to send to blockchain.

def commit\_transaction(self): - aborts transactions and initiates sending of all data to blockhain.

def abort\_transaction(self): - only proxy to mongodb