ChainBridge

Axel Ország-Krisz Dr.   Richárd Ádám Vécsey Dr.

Table of Contents

I. Summary

1. ChainBridge

ChainBridge is a service and application to transform BCH transaction history into old-school government, banking or credit-scoring related documents.

We called our solution a bridge, since we want to make a connection between the new, digital assets and the traditional banking habits and approaches. The COVID-19 epidemic shows, that cryptocurrencies can be good alternatives to classic fiat money transactions. However, people, who handle confident their own money or bank account, can struggle with the field of cryptocurrencies. Different definitions and strange words make barriers to wider usage of digital assets. Our solution helps people to handle their own cryptocurrency account as a common bank account with familiar services such as creating Statement of Account or getting Account Activity history. Filter functions and statistics help to follow up incoming and outgoing transactions as well. Users can create well-formatted and organized documents for further usage such as certificate to the local tax authority or appendix of a contract that is made away from keyboard. Our documents can be accessed online by their unique identification, this protects the system against faking. Users can hide the actual wallet ID from authorities or other entities while having a certificate, which proves the shown data is based on blockchain information.

We believe, that cryptocurrency is not only for technocrats, hipsters or IT gurus, but for the owner of a local grocery store, taxi driver or the old lady from the neighborhood.

2. Vision

Our vision is to establish a service that helps people to get actively involved in the world of cryptocurrencies with daily spending and transactions while using and monitoring their own wallet just the same way as the ordinary bank account during online banking; with the same methods, words and concepts.

3. Description of the problem

Based on different global surveys such as Deloitte’s 2020 Global Blockchain Survey or PwC’s Global Blockchain Survey about blockchain we can state there is still lack of trust against blockchains due to various reasons. Key arguments relate to implementation, regulation, compliance and the lack of trust in general. However blockchain technology itself promises and realizes a high level of transparency, understanding and experiencing this attribute of that technology is not that obvious, since the use of the existing high quality blockchain tools mostly requires some level of knowledge about the blockchain technology itself. Giving a simple example, comparing the view of a bank account history to the view of the history of a blockchain wallet, seems like we would order extra parts to our new vehicles directly with product numbers instead of their names.

The character of blockchain transactions and banking transactions is very similar and very different at the same time. Banking transaction highly tries to demonstrate privacy at the access of the data but is very traditional and obvious from the view of the data itself. Blockchains on the contrary are mostly publicly accessible but the form of transaction data need to be further processed.

Irrespectively of our wish to live in total freedom or in a high-tech world only, reality affects our lives. We have to pay the taxes we have to manage our entrepreneurship or work and of course our daily life. Some of those circumstances don't let us enjoy the advantages of cutting edge technologies like blockchain. Even though those circumstances seem to be way too old-school, they are important and they have their own nature. We, developers of cutting edge technologies have to build tools for those parts of the world to let them close up quickly.

II. The Business Conception

In this chapter there are the most important parts of our business idea with some details and explanation. The core parts of this business idea are the business model canvas, the provided services, the accessibility of ChainBridge, the use-cases and the time frame of business planning.

1. Provided services

Our solution contains the following services. **Highlighted** names mean the service will be provided instantly at the start. Any other services will be provided later in the future

- **Statement of Account**
- **Account Activity**
- **Certification of Account**
- **Certification of Activities**
- **Details for Certifications**
- **Assistance for our Services**
- Tax Authority Compatible Files
- Preconfigured Filters for Tax Calculation
- Assistance for Tax Calculation and Taxation
- Notifications via Email
- Extra Secured Connection and Queries with our own Blockchain Explorer Service

Statement of Account is created in a certain time range, such as monthly or yearly. This is similar to the classic statement of account in the banking sector. Since blockchain contains extra information, that are not exist in the financial sector, there are optionally statistics fields on the statements. It contains two big parts such as Account Summary and Account Activity. The balance and values are available in BCH and in fiat currency, such as USD, EUR, GBP, etc. **Account Activity** is a query and filter based service. The result of Account Activity without any filter on a unique month is equals with the content of Account Activity in the Statement of Account on the same month. Account Activity helps to browse transaction history for any specific reason for example understanding the spending, checking the transaction from a distinct wallet or creating transaction history in a bigger time frame.

This is the schematic and not comprehensive list about data that we provide in our core service based on blockchain:

- Account Summary
    - General data
        - Account Address
    - Balance
        - Beginning Balance
        - Total Received (Inputs)
        - Net Sent (Outputs)
        - Fees
        - Total Sent (Outputs + Fees)
        - Ending Balance
    - Statistics
        - Number of affected blocks
        - Number of affected transactions
        - Number of deposits
        - Number of withdrawals
        - Number of unique transactions

- Account Activity
  - Transactions
    - Account Address
    - Filter description
    - Date
    - Foreign address
    - Name / nick / tag for foreign address
    - Amount
    - Fees
    - Block height
    - Transaction ID
    - Number of Confirmations
  - Summary line

It is possible that anyone does not want to reveal their wallet address or just want to filter special transactions from blockchain. We provide a certification service for them. Certificates can be made as **Certification of Account** or **Certification of Activities**. These outputs are similar with the output of the first two service. However, the biggest difference, that the data on document may be less and we provide extra credibility for documents by certification. With a unique certification ID, anybody can check the validity of the document on our server as part of the **Details for Certifications** service. When our user share the unique ID of document, their client, Tax Authority or any 3rd party partner has the chance to check the validation of the given document. **Assistance** means a helpdesk service. There are a lot of question marks can be pop up in our clients mind when they want to jump from an old, paper-based world into a cutting edge universe. We want to serve and educate our clients at the same time. That's why it is necessary to have chat bots and a human-based helpdesk service.

After the starting our core services, we want to create new ones to provide better user experience and solve more real life problems. Since one of our use-case connects to taxation, we want to increase the level of service with **Tax Authority Compatible Files**. These documents will be based on the exact needs of tax authorities. To achieve this level, we plan to begin a discourse with different authorities around the world. In this moment, the user has the necessary input and output data for taxation, but only one thing missing: the exact amount of the tax to pay. In **Preconfigured Filters for Tax Calculation** service the user can calculate the tax with just some clicks. Automated processes get outsource data for example exchange rates to the moment in transactions. However, the process deeply automated, user always has the right and possibility to check, override or overrule the system. We want to handle non-technocrat users as well. Therefore it will be necessary to provide **Assistance for Tax Calculation and Taxation**. Chatbots and classic human helpdesk can navigate he user through the taxation process. It is important to emphasize, we are not tax office, the helpdesk can provide technical support about technology and our services. This is important, since there is a little chance that a clerk from local tax authority is familiar with blockchain based assets. They know the rules and regulation that share with our user, and we know the technology background.

As most bank send statement of account periodically for their customers, this service cannot be missing from the cryptocurrency wallet. **Notifications via Email** is an eco-friendly way to provide automated documents periodically. Users should not give up their old habits to monitor their own transactions month by month. Furthermore, we can send documents based on preset triggers, for example when a transaction from a specified account is happened.

Some user may need extra security due to various reasons. **Extra Secured Connection and Queries with our own Blockchain Explorer Service** provides this safety. The searches of user won't be executed on a third party blockchain explorer via API. Our service will be able handle all queries without any outsource service. We plan to run a onion domain as well to serve people from Tor network. There are a lot of country or job where people had to use Tor to defend their own identity, for example journalist and their informants in a hostile country with political repression by the regime. Extra services respect that need of potential users.

2. Use cases

Building extra credibility based on BCH transaction

This is a very common use-case. Renting a house or a car, buying expensive electronic devices for loan, especially mobile phone for any “buy now, pay later” financial construction, are regularly transaction in the life. Nowadays the banking sector is in a monopolized situation, since only banking documents can prove the credibility of customer. For example when someone pays periodically for a service in a fiat money from their own bank account, they can save a transaction history or just use some certification of account to prove the number and periodicity of those money transfers. It is possible to use a normal blockchain explorer for this task. However, the format of output is very different than documents from a financial institute.

Imagine a situation when a cool entrepreneur want to make a great deal. The task is to show to the other partner this coolness. If this entrepreneur makes financial transaction in a huge value, documents can confirm that the business model is consistent with the spoken words. It would be great if they could be able to use their own wallet history to represent the financial background and financial habits.

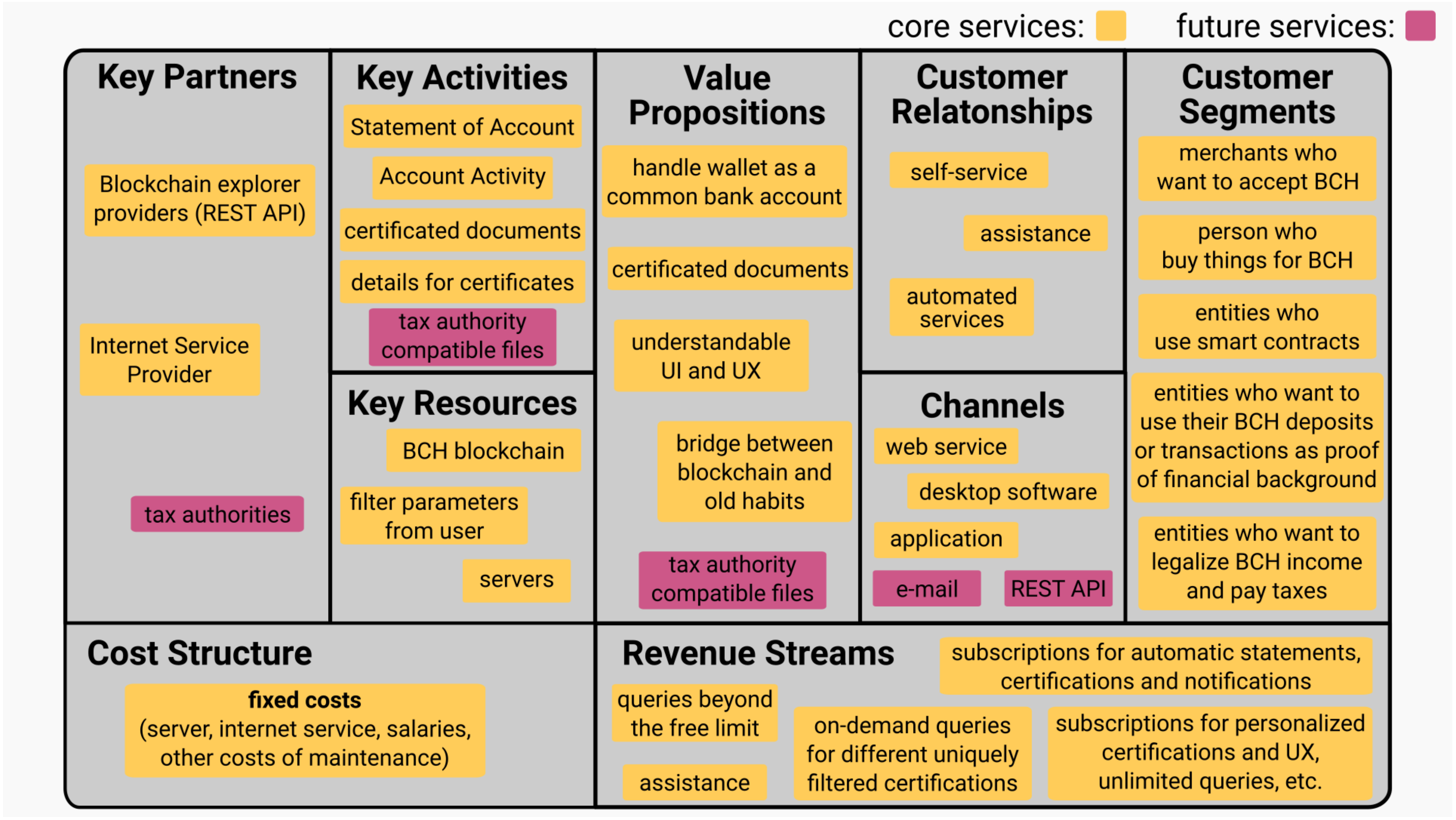
Renting a new flat or house in most countries of Europe can be a nightmare, since the possible renter should certify two things: their earlier renting history and that every transaction have done on the time month by month. However, documents from a bank contains a lot of personal and sensitive data that should be handled properly based on a very strict data protection rules. The financial sector sees the problem and there is a new possibility to give permission to anyone for getting a slice of the full transaction history without any personal or sensitive data. As this example shows, banks are not old, slow and based their daily operation on old habits. For spreading worldwide the blockchain based lifestyle, technology and services should give solutions for these simple problems of adult life. Therefore to transform the blockchain explorers output data to a more human readable form is essential.

Paying taxes on BCH transactions

Taxation is an enormous barrier for using BCH transaction in commerce or e-commerce. In general the tax authorities and the tax laws can handle the taxation problems of cryptocurrency. For example IRS issued IRS Notice 2014-21, IRB 2014-16 to help the taxation of virtual currencies. This guide separates different use cases such as trading, business life and holding digital assets as capital asset. In spite of the rules, taxpayers should have documents that are the base of tax calculation. A document based on a common bank account is always suited for the requirements. In a lot of countries tax authorities can directly see the cash flow of companies in one way or another, for example via reports of online cash registers. Cryptocurrencies are slightly different. To widen the acceptance of digital assets, we should provide documents for companies and customers that suit for the needs of taxation and at least are equivalent with the banking documents. However, we want to begin a dialogue with tax authorities to create well formatted documents that helps to reduce the time and stress-factor of taxation.

Checking validity of certified documents

Our user can chose certified documents as output. Each document contains a unique ID number. With this number anybody can check the validity of our certification. It helps to add more credibility for the document and for our user. The user has the chance to personalize the content of a certified document, for example hiding any data. It is different than a normal Activity Account filtering, since in those service, user can select from the transaction based on the filters any has a very limited flexibility to hide or reveal cells on the output document. In a certification service the user can use only aliases instead of remote address, hide any wallet addresses including their own address, hide the information about blockchain or time. This flexible solution fits for any needs that can occur. This service create balance between providing information with credibility and save privacy data. Any third party person can get information that they has the right to handle. A properly created document reduces the risk rejection by authorities based on any data protection rules. If a stakeholder has not the right to handle a distinct data, user can hide it in the document. The certification prove that the content of document was the part of the blockchain at the moment of creation.


core services: 🟨 future services: 🟥

Key Partners
- Blockchain explorer providers (REST API)
- Internet Service Provider
- tax authorities

Key Activities
- Statement of Account
- Account Activity
- certificated documents
- details for certificates
- tax authority compatible files

Key Resources
- BCH blockchain
- filter parameters from user
- servers

Value Propositions
- handle wallet as a common bank account
- certificated documents
- understandable UI and UX
- bridge between blockchain and old habits
- tax authority compatible files

Customer Relatonships
- self-service
- assistance
- automated services

Channels
- web service
- desktop software
- application
- e-mail
- REST API

Customer Segments
- merchants who want to accept BCH
- person who buy things for BCH
- entities who use smart contracts
- entities who want to use their BCH deposits or transactions as proof of financial background
- entities who want to legalize BCH income and pay taxes

Cost Structure
- **fixed costs** (server, internet service, salaries, other costs of maintenance)

Revenue Streams
- subscriptions for automatic statements, certifications and notifications
- queries beyond the free limit
- on-demand queries for different uniquely filtered certifications
- subscriptions for personalized certifications and UX, unlimited queries, etc.
- assistance

Key Partners

- Blockchain explorer providers (REST API)
- Internet Service Provider
- Tax Authorities

Key Activities

- creating Statement of Account
- creating Account Activity
- creating Certification of Account
- creating Certification of Activities
- providing details for certificates
- creating tax-authorities-compatible files

Key Resources

- BCH blockchain
- filter parameters from user
- servers
- forms and regulations from different tax authorities

Value Propositions

- handle wallet as a common bank account
- certificated documents
- understandable UI and UX
- bridge between blockchain and old habits
- tax-authorities-compatible files

Customer Relationships

- self-service
- automated-service
- assistance

Channels

- web service
- desktop software
- application
- e-mail
- REST API

Customer Segments

- merchants who want to accept BCH
- person who buy things for BCH
- entities who use smart contracts
- entities who want to use their BCH deposits or transactions as proof of their financial background
- entities who want to legalize BCH income and pay taxes

Cost Structure

- fixed costs: server, internet service, salaries, other costs of maintenance

Revenue Streams

- on-demand queries for different uniquely filtered certifications
- longer existing certifications
- subscriptions (monthly, yearly) for automatic statements, certifications and notifications
- subscriptions (monthly, yearly) for personalized certifications and user experience, unlimited queries, stored history on server
- queries beyond the free limit
- assistance

4. Timing

months	platform side	other process	started services
present	● web service ● desktop software with GUI ● command line software ● supporting English ● supporting Spanish ● supporting German	● early marketing campaign	● Statement of Account ● Account Activity ● Certification of Account ● Certification of Activities ● Assistance
3	● Android application ● iOS application ● supporting French	● cooperate with tax authorities for better formatted files	● FAQ
6	● chatbot assitant	● cooperate with tax authorities for better formatted files	● chatbot assitant
9	● supporting Afrikaans	● continue the cooperation with tax authorities ● cooperate with financial institutes for credit-scoring compatible files ● middle marketing campaign	● tax authority compatible files ● preconfigured filters for tax calculation ● assistance for tax calculation and taxation
12	● e-mail service ● supporting Portuguese	● continue the cooperation with financial institutes	● notifications via e-mail
15	● ASR + TTS modules	● providing equal opportunities for visually imparied people without human assistance service	● ASR + TTS assistant
18	● providing REST API ● supporting Polish	● late-term marketing campaign	● API service
21	● own blockchain explorer ● .onion access point ● supporting more languages	● analyze user habits and feedbacks for determining future developments	● extra secured connection and queries with our own blockchain explorer ● services are available via .onion (Tor) network
24			

5. Establishing services and platforms

	months	0-3	3-6	6-9	9-12	12-15	15-18	18-21	21-24
type	name								
platforms	Web service	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪
	Desktop software with GUI	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪
	Command line program	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪
	Android application	⬛	🟪	🟪	🟪	🟪	🟪	🟪	🟪
	IOS application	⬛	🟪	🟪	🟪	🟪	🟪	🟪	🟪
	Email	⬛	⬛	⬛	🟪	🟪	🟪	🟪	🟪
	REST API	⬛	⬛	⬛	⬛	⬛	🟪	🟪	🟪
	.onion (Tor) network	⬛	⬛	⬛	⬛	⬛	⬛	⬛	🟪
help	Assistance	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
	FAQ	⬛	🟨	🟨	🟨	🟨	🟨	🟨	🟨
	Chatbot	⬛	⬛	🟨	🟨	🟨	🟨	🟨	🟨
	ASR + TTS	⬛	⬛	⬛	⬛	🟨	🟨	🟨	🟨
services	Statement of Account	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	Account Activity	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	Certification of Account	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	Certification of Activities	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	Details for Certifications	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	Assistance for our Services	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	Tax Authority Compatible Files	⬛	⬛	⬛	🟩	🟩	🟩	🟩	🟩
	Configured Filters for Tax Calculation	⬛	⬛	⬛	🟩	🟩	🟩	🟩	🟩
	Assistance for Tax Calculation, Taxation	⬛	⬛	⬛	🟩	🟩	🟩	🟩	🟩
	Notifications via Email	⬛	⬛	⬛	⬛	🟩	🟩	🟩	🟩
	Extra Secured Connection and Queries	⬛	⬛	⬛	⬛	⬛	⬛	⬛	🟩
languages	English	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
	Spanish	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
	German	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
	French	⬛	🟥	🟥	🟥	🟥	🟥	🟥	🟥
	Afrikaans	⬛	⬛	⬛	🟥	🟥	🟥	🟥	🟥
	Portuguese	⬛	⬛	⬛	⬛	🟥	🟥	🟥	🟥
	Polish	⬛	⬛	⬛	⬛	⬛	⬛	🟥	🟥
	Others	⬛	⬛	⬛	⬛	⬛	⬛	⬛	🟥

III. Under the hood

This chapter contains information about the code and structure of ChainBridge.

1. Data

Data sources:

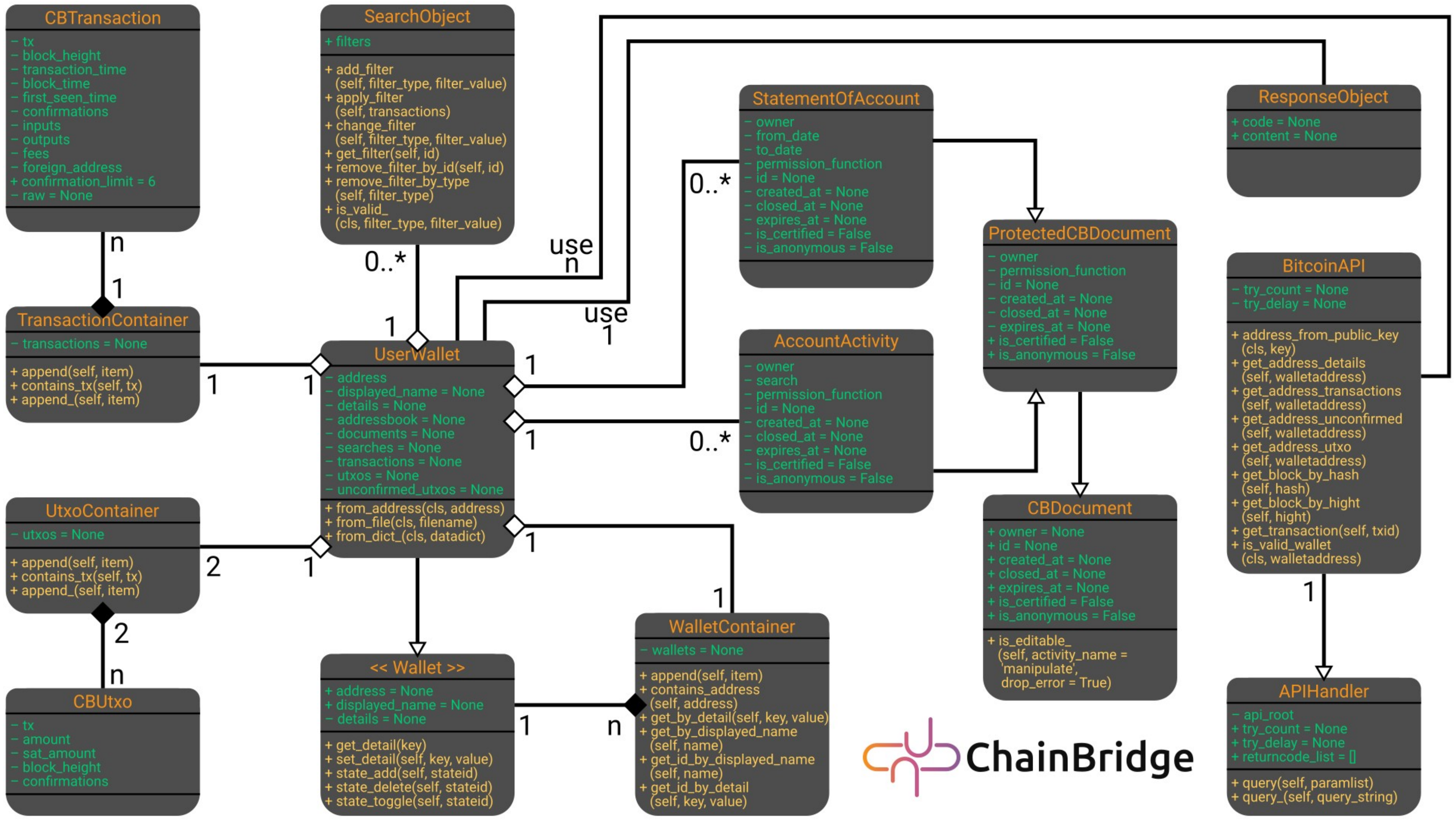
- Blockchain
- User
- Own server (stored certifications)

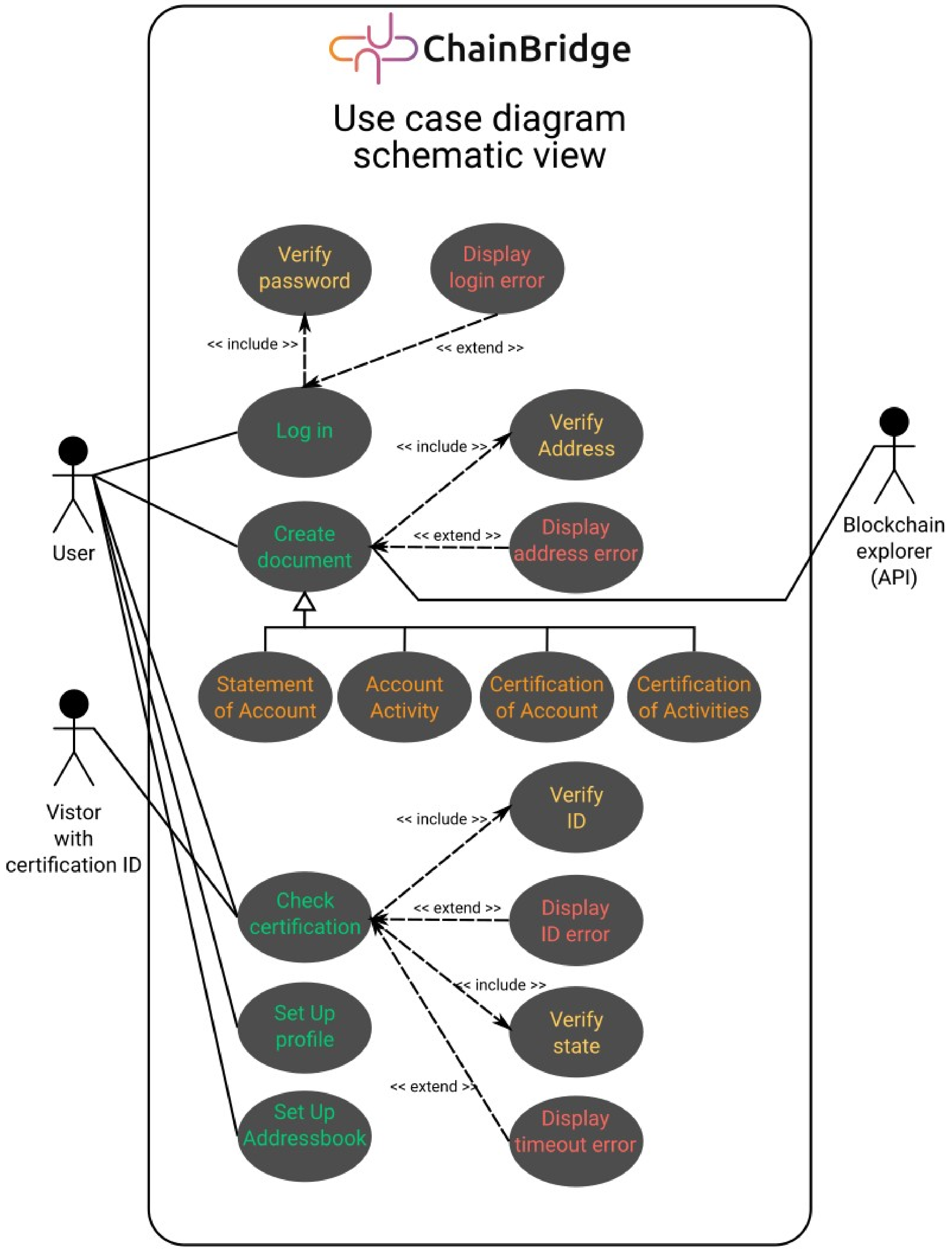
State of Documents:

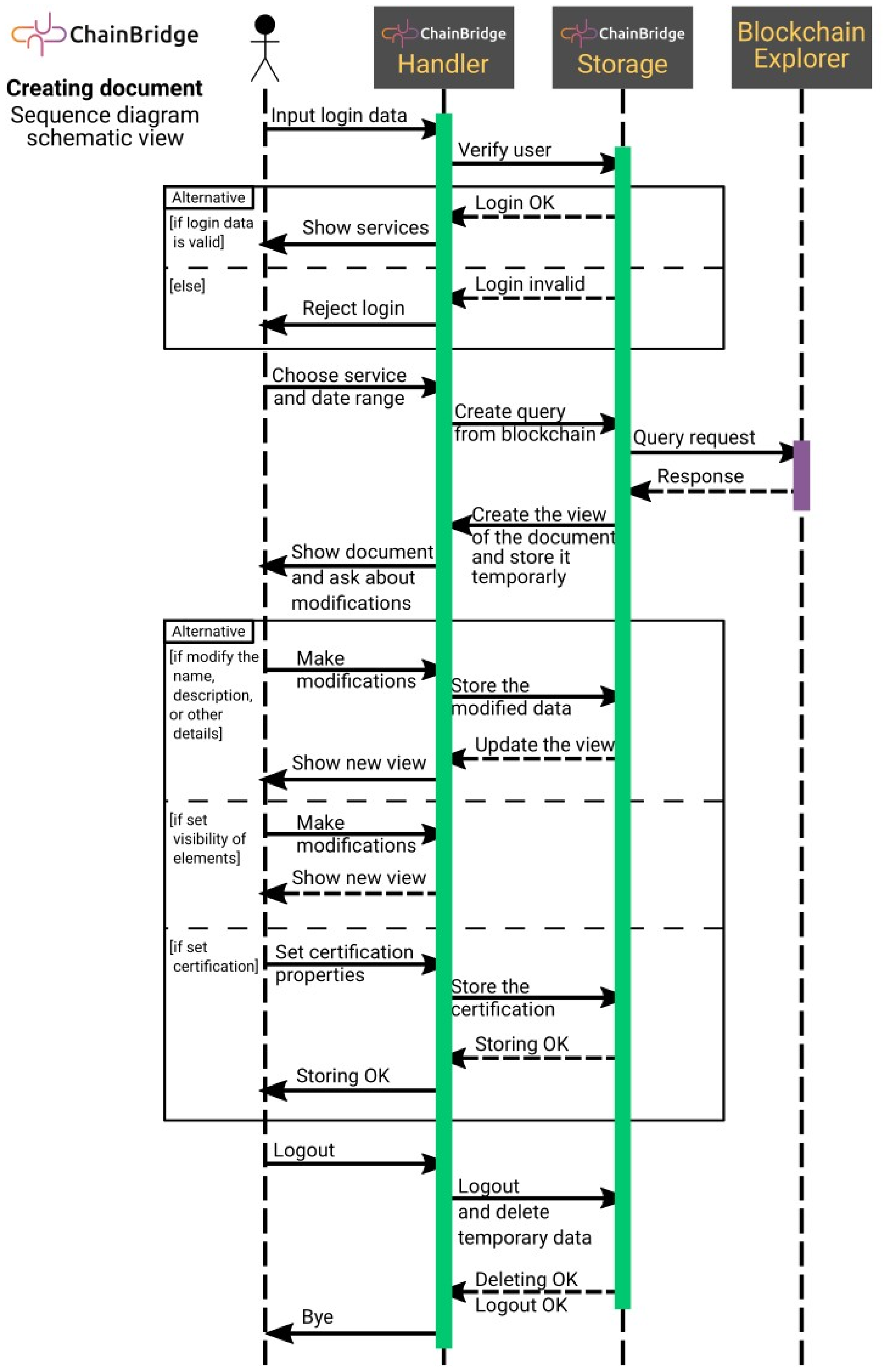
- instantiated
- created
- has_id
- closed
- expires
- is_anonymous
- is_certified
- expired

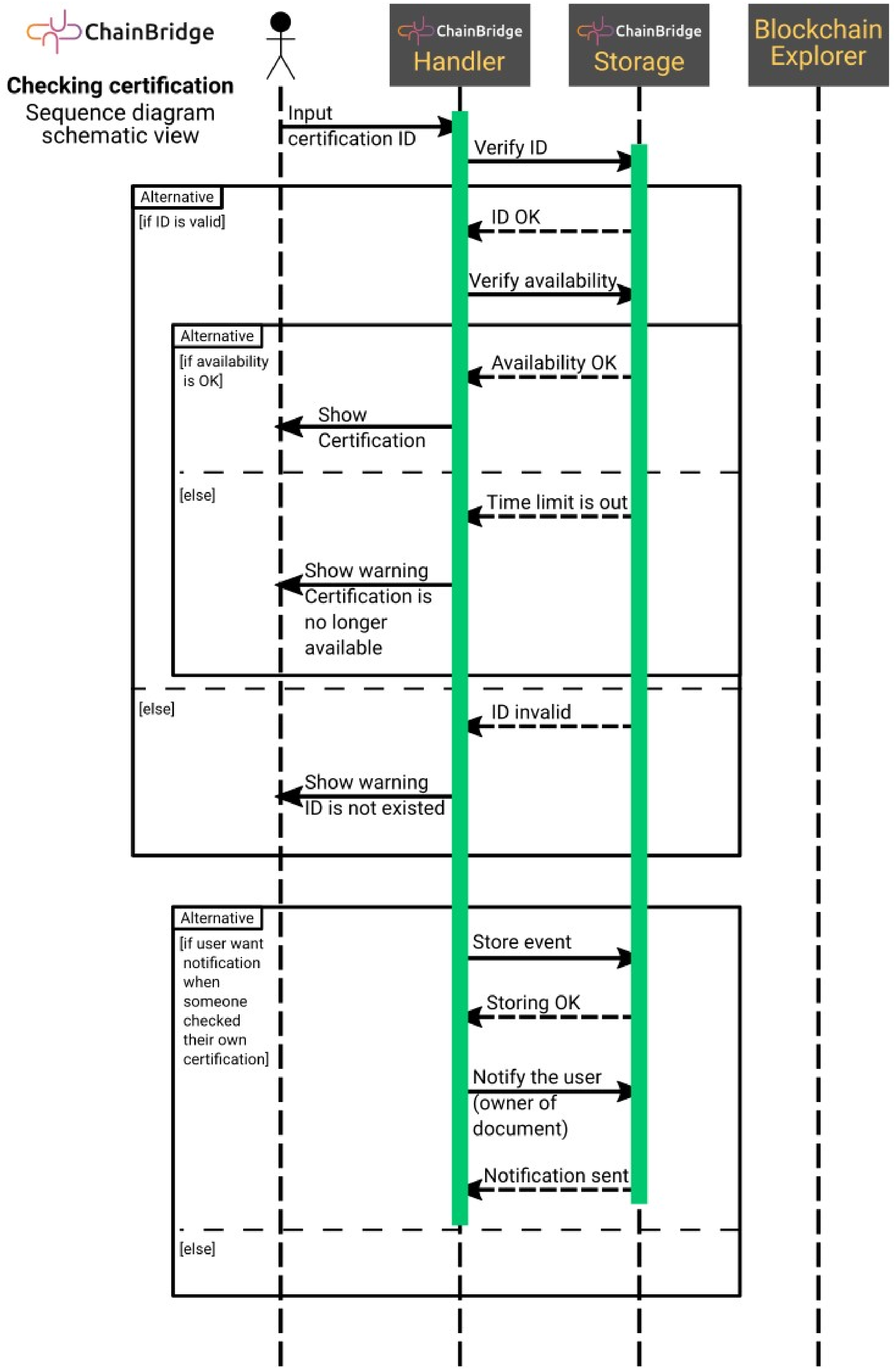
2. UML

Under the UML section we provide a class diagram, a use case diagram, and two sequence diagrams. One is for a scenario called **creating a document** and the other is for the **checking certification** scenario. Diagrams are created in a schematics view, since there are no enough space in pages of this book or on a FullHD screen to show every decision and all of parts at the same time.


CBTransaction
- tx
- block_height
- transaction_time
- block_time
- first_seen_time
- confirmations
- inputs
- outputs
- fees
- foreign_address
+ confirmation_limit = 6
- raw = None
n
1
TransactionContainer
- transactions = None
+ append(self, item)
+ contains_tx(self, tx)
+ append_(self, item)
1
1
SearchObject
+ filters
+ add_filter
(self, filter_type, filter_value)
+ apply_filter
(self, transactions)
+ change_filter
(self, filter_type, filter_value)
+ get_filter(self, id)
+ remove_filter_by_id(self, id)
+ remove_filter_by_type
(self, filter_type)
+ is_valid_
(cls, filter_type, filter_value)
0..*
1
UserWallet
- address
- displayed_name = None
- details = None
- addressbook = None
- documents = None
- searches = None
- transactions = None
- utxos = None
- unconfirmed_utxos = None
+ from_address(cls, address)
+ from_file(cls, filename)
+ from_dict_(cls, datadict)
use
n
use
1
StatementOfAccount
- owner
- from_date
- to_date
- permission_function
- id = None
- created_at = None
- closed_at = None
- expires_at = None
- is_certified = False
- is_anonymous = False
0..*
1
AccountActivity
- owner
- search
- permission_function
- id = None
- created_at = None
- closed_at = None
- expires_at = None
- is_certified = False
- is_anonymous = False
1
0..*
ResponseObject
+ code = None
+ content = None
ProtectedCBDocument
- owner
- permission_function
- id = None
- created_at = None
- closed_at = None
- expires_at = None
+ is_certified = False
+ is_anonymous = False
CBDocument
+ owner = None
+ id = None
+ created_at = None
+ closed_at = None
+ expires_at = None
+ is_certified = False
+ is_anonymous = False
+ is_editable_
(self, activity_name =
'manipulate',
drop_error = True)
BitcoinAPI
- try_count = None
- try_delay = None
+ address_from_public_key
(cls, key)
+ get_address_details
(self, walletaddress)
+ get_address_transactions
(self, walletaddress)
+ get_address_unconfirmed
(self, walletaddress)
+ get_address_utxo
(self, walletaddress)
+ get_block_by_hash
(self, hash)
+ get_block_by_hight
(self, hight)
+ get_transaction(self, txid)
+ is_valid_wallet
(cls, walletaddress)
1
APIHandler
- api_root
+ try_count = None
+ try_delay = None
+ returncode_list = []
+ query(self, paramlist)
+ query_(self, query_string)
UtxoContainer
- utxos = None
+ append(self, item)
+ contains_tx(self, tx)
+ append_(self, item)
2
1
2
n
CBUtxo
- tx
- amount
- sat_amount
- block_height
- confirmations
1
1
1
<< Wallet >>
+ address = None
+ displayed_name = None
- details = None
+ get_detail(key)
+ set_detail(self, key, value)
+ state_add(self, stateid)
+ state_delete(self, stateid)
+ state_toggle(self, stateid)
1
n
WalletContainer
- wallets = None
+ append(self, item)
+ contains_address
(self, address)
+ get_by_detail(self, key, value)
+ get_by_displayed_name
(self, name)
+ get_id_by_displayed_name
(self, name)
+ get_id_by_detail
(self, key, value)
ChainBridge


ChainBridge
Use case diagram schematic view
Verify password
Display login error
<< include >>
<< extend >>
Log in
User
Verify Address
<< include >>
Create document
<< extend >>
Display address error
Blockchain explorer (API)
Statement of Account
Account Activity
Certification of Account
Certification of Activities
Vistor with certification ID
Verify ID
<< include >>
Check certification
<< extend >>
Display ID error
<< include >>
Verify state
Set Up profile
<< extend >>
Set Up Addressbook
Display timeout error


ChainBridge
Creating document
Sequence diagram
schematic view
ChainBridge Handler
ChainBridge Storage
Blockchain Explorer
Input login data
Verify user
Alternative
[if login data is valid]
Login OK
Show services
[else]
Login invalid
Reject login
Choose service and date range
Create query from blockchain
Query request
Response
Create the view of the document and store it temporary
Show document and ask about modifications
Alternative
[if modify the name, description, or other details]
Make modifications
Store the modified data
Update the view
Show new view
[if set visibility of elements]
Make modifications
Show new view
[if set certification]
Set certification properties
Store the certification
Storing OK
Storing OK
Logout
Logout and delete temporary data
Deleting OK
Logout OK
Bye


ChainBridge
Checking certification
Sequence diagram
schematic view
ChainBridge
Handler
ChainBridge
Storage
Blockchain
Explorer
Input
certification ID
Verify ID
Alternative
[if ID is valid]
ID OK
Verify availability
Alternative
[if availability
is OK]
Availability OK
Show
Certification
[else]
Time limit is out
Show warning
Certification is
no longer
available
[else]
ID invalid
Show warning
ID is not existed
Alternative
[if user want
notification
when
someone
checked
their own
certification]
Store event
Storing OK
Notify the user
(owner of
document)
Notification sent
[else]

3. Code reference

class ResponseObject(object):

This class holds response data of an API query

def __init__(self, code=None, content=None):

Initializes the ResponseObject

```
==============================

Parameters

----------
```

code: int, optional (None if omitted)

The return code of a response. Having a return code of 200 usually means that everything went just fine. Having a return code different than 200 are usually proof of error.

content: dict, list, optional (None if omitted)

The returned data of a response. Common usage of this object can be sending of JSON based data which is transformed into a structured Python variable. Variables deserialized from JSON used to be a mix of list and dict instances.

```
Attributes

----------

code

content
```

```
@property
def code(self):

    Returns the status code of a request

    ====================================

    Returns
    -------

    int, None

    Status code, if already not set, the status code is none.


@code.setter
def code(self, newcode):

    Sets the request's status code

    ==============================

    Parameters
    ----------

    newcode: int

    Code to set as response status code. This value can be set at
    once only on an instance.

    Throws
    ------

    PermissionError

    If tried to set a response code when it is different than
    None.
```

```
@property
def content(self):
    Returns the content of the response
    ===================================
    Returns
    -------
    any
    The content of the response. The content itself used to be a
    mix of lists and dictionaries since JSON is a quite common
    response format and Python interprets those string into types
    mentioned above.


@content.setter
def content(self, newcontent):
    Sets the content of the response
    ================================
    Parameters
    ----------
    newcontent: any
    The content of the request's response. It can be any type, bit
    a mix of list and dict instances are quite common in case of
    API services, because they are using JSON to serialize data.
```

class APIHandler(object):

This class helps to manage SEO style REST API queries in a convinient way

def __init__(self, api_root, try_count=None, try_delay=None, returncode_list=[]):

Initializes APIHandler object

=============================

Parameters

----

api_root: string

The URL base of the future API queries. Due to the habit of the Python's request library, http://, https:// or other supported qualifiers couldn't be omitted.

try_count: int, optional (None if omitted)

Number of tries to have a sucessful query. If the value of this parameter omitted the value of DEFAULT_TRY_COUNT is used.

try_delay: int, optional (None if omitted)

Number of milliseconds to have a new try with the same query. If the value of this parameter is omitted the value of DEFAULT_TRY_DELAY is used instead.

returncode_list: list of int, optional (empty list if omitted)

This parameter lets the user to select error codes to accept, send back. By default request have result in case only, when return code from the server is 200. Adding elements to this parameter will cause that some error codes are sent back as result.

@property

def api_root(self):

    Returns the Api root

    =====================

    string

    The API root.

def query(self, paramlist):

    Makes a query from the API

    ==========================

    Parameters

    ----

    paramlist: list of strings

    The content of paramlist is transformed into SEO friendly API query string. This means elements of the list gets joined together with a slash seperator.

    Returns

    ResponseObject

    If the query is successful a full ResponseObject instance is returned with content.

    ResponseObject (.content=errorcode; .content=None)

    If there is a list of acceptable error codes and the query ends with that error code, the returned ResponseObject has

```
    a .content attribute with None value and a code attribute with
    the received error code.

    None

    In case of total unsuccess the value of the return is None.

    Notes
    -----
    This method is a wrapper of the .query_() method. For common
    use cases the use of this method is preferred because of its
    user friendlynes.

    See Also
    --------
    To have more impression about the result of this method please

    consult the docs of the .query_() method too.
```

@property

def returncode_list(self):

```
    Gets the list of acceptable return codes
    ========================================
    Returns
    -------
    list of int

    List of acceptable error codes given by the user.

    Notes
    -----
    However 200 is an accepted return code it shouldn't be pert of
    that list because it's the default return code for success.
```

@returncode_list.setter

def returncode_list(self, newlist):

```
Sets the list of acceptable return codes

========================================

Parameters

----------
```

newlist : list of int

```
Replaces the list of acceptable return codes.

Notes

-----

1.

200 is always accepted as this is the return code of success.

2.

In case if the modification of the returncode_list is
temporary, the original value should be saved and reset.

Example

-------

# To check whether a page exists or not the following example
is

# quite simple.

handler = APIHandler('example.com')

# ...

# Check if page exists

old_list = handler.returncode_list
```

```
    handler.returncode_list = [404]

    data = handler.query(['example'])

    if data is not None:

    if data.code === 200:

    print('Page exists.')

    if data.code == 404:

    print('Page doesn\'t exist.')

    else:

    print('Other error happen.')

    handler.returncode_list = old_list
```

@property

def try_count(self):

```
    Gets the amount of tries made in case of unsuccess

    ==================================================

    Returns

    -------

    int

    Number of tries.
```

```
@try_count.setter
def try_count(self, try_count):

    Sets the number of tries in case of unsuccess

    =============================================

    Parameters

    ----------

    try_count: int

    The number of total tries in case of unsuccess.

@property
def try_delay(self):

    Gets the delay between requests in case of unsuccess

    ====================================================

    Returns

    -------

    int

    Delay in milliseconds.

@try_delay.setter
def try_delay(self, try_delay):

Sets the delay between requests in case of unsuccess

====================================================

    Parameters

    ----------

    try_delay: int

    Delay in milliseconds.
```

def query_(self, query_string):

```
Processes an actual query
=========================
Parameters
----------
```

query_string: str

```
String to be added to the API root to make a query.
Returns
-------
ResponseObject
```

If the query is successful a full ResponseObject instance is returned with content.

ResponseObject (.content=errorcode; .content=None)

If there is a list of acceptable error codes and the query ends with that error code, the returned ResponseObject hes a .content attribute with None value and a code attribute with the received error code.

None

In case of total unsuccess the value of the return is None.

```
Notes
-----
```

The use of .query() method is preferred.

```
See Also
--------
```

Please consult the documentation for .query() before the use of this method.

class BitcoinAPI(APIHandler):

```
This class provides simple interface to bitcoin.com REST API
=============================================================
Notes
-----
This object doesn't implement all the functionality of the
bitcoin.com blockhain explorer API. It implements only those,
which are needed to serve the purposes of ChainBridge.
See Also
--------
Documentation of the bitcoin.com REST API can be consulet at:
https://rest.bitcoin.com/
```

def __init__(self, try_count=None, try_delay=None):

```
Initializes the BitcoinAPI object
=================================
Parameters
----------
```

try_count: int, optional (None if omitted)

```
Number of tries to have a sucessful query. If the value of
this parameter omitted the value of DEFAULT_TRY_COUNT is used.
```

try_delay: int, optional (None if omitted)

```
Number of milliseconds to have a new try with the same query.
If the value of this parameter is omitted the value of
DEFAULT_TRY_DELAY is used instead.
Classmethods
------------
address_from_public_key
is_valid_wallet
```

```
@classmethod
def address_from_public_key(cls, key):
    Gets bitcoincash address from public key
    ========================================
    Parameters
    ----------
    key: str
    The string of a known public key to query.
    Returns
    -------
    str
    The bitcoincash address of the wallet.
    None
    If there is no address behind the public key or if the query
    wasn't successful at all.
    Notice
    ------
    This is a classmethod you can call it without instantiating a
    BitcoinAPI object.
    Example
    -------
    # Get the address that belongs to a known public key.
    data = BitcoinAPI.address_from_public_key('XXX')
    if data is not None:
    print('The address is: {}'.format(data))
    else:
    print('This key doesn\'t lead to a valid address or some error
    happened.')
```

```
def get_address_details(self, walletaddress):

    Gets the details record of the address

    ======================================

    Parameters

    ----------

    walletaddress : str

    The bitcoincash address of the wallet.

    Returns

    -------

    dict

    The details record that belongs to the given wallet.

    Throws

    ------

    ValueError

    If the given address is not valid.

    Notes

    -----

    The service works with legacy and SLP addresses as well but
    this code prefers to use bitcoincash address where possible.
```

```
def get_address_transactions(self, walletaddress):

    Gets transaction records of the address

    =======================================

    Parameters

    ----------

    walletaddress: str

    The bitcoincash address of the wallet.

    Returns

    -------

    dict

    List of all transactions that belongs to the given wallet.

    Throws

    ------

    ValueError

    If the given address is not valid.

    RuntimeError

    If the address gets invalid during pagination of the

    transaction list.

    RuntimeError

    If error, other than invalid user occures during the

    pagination of

    the transaction list.

    Notes

    -----

    The service works with legacy and SLP addresses as well but

    this code prefers to use bitcoincash address where possible.
```

```
def get_address_unconfirmed(self, walletaddress):

    Gets the list of unconfirmed utxos of the address

    =================================================

    Parameters

    ----------

    walletaddress: str

    The bitcoincash address of the wallet.

    Returns

    -------

    dict

    The list of unconfirmed utxos that belongs to the given
    wallet.

    Throws

    ------

    ValueError

    If the given address is not valid.

    Notes

    -----
```

```
def get_address_utxo(self, walletaddress):

    Gets the list of confirmed utxos of the address

    ==============================================

    Parameters

    ----------

    walletaddress: str

    The bitcoincash address of the wallet.

    Returns

    -------

    dict

    The list of confirmed utxos that belongs to the given wallet.

    Throws

    ------

    ValueError

    If the given address is not valid.

    Notes

    -----

    The service works with legacy and SLP addresses as well but
    this code prefers to use bitcoincash address where possible.
```

```
def get_block_by_hash(self, hash):

    Gets the block data with the given hash

    =======================================

    Parameters

    ----------

    hash: str

    The hash of the bitcoincash block.

    Returns

    -------

    dict

    The full record of the block.

    Throws

    ------

    ValueError

    If the given hash is not valid.


def get_block_by_hight(self, hight):

    Gets the block data with the given height

    ========================================

    Parameters

    ----------

    height: int, str

    The height of the bitcoincash block.

    Returns
```

```
-------

dict

The full record of the block.

Throws

------

ValueError

If the given height is not valid.
```

def get_transaction(self, txid):

```
Gets the transaction data with the given txid

=============================================

Parameters

----------
```

txid: str

```
The txid of the bitcoincash transaction.

Returns

-------

dict

The full record of the transaction.

Throws

------

ValueError

If the given txid is not valid.
```

```
@classmethod
def is_valid_wallet(cls, walletaddress):
    Checks whether the address is valid or not
    ==========================================
    Parameters
    ----------
    walletaddress: str
    The bitcoincash address of the wallet.
    Returns
    -------
    bool
    True if the address is valid, False if not or if something
    error happen.
    Notes
    -----
    1.
    This is a classmethod you can call it without instantiating a
    BitcoinAPI object.
    2.
    The service works with legacy and SLP addresses as well but
    this code prefers to use bitcoincash address where possible.
    Example
    -------
    # Check a bitcoincash address
    if BitcoinAPI.is_valid_wallet('bitcoincash:XXX'):
    print('This is a valid address.')
    else:
    print('This address seems to be non-valid.')
```

class Wallet(object):

This class provides basic functionality of a wallet

def __init__(self, address=None, displayed_name=None, details=None):

```
Initializes the Wallet object
=============================
Parameters
----------
```

address: str, optional (None if omitted)

The address of the wallet instance.

displayed_name: str, optional (None if omitted)

The name to display for the wallet.

details: dict, optional (None if omitted)

Key, values pairs (str, str) of information about the wallet.

```
Attributes
----------
address
details
displayed_name
state
```

```
Class Level Constants
---------------------
WALLET_INSTANTIATED
WALLET_VALID
WALLET_EDITABLE
```

```
@property
def address(self):

    Gets the address of the wallet

    ==============================

    Returns:

    str

    The address of the wallet.


@address.setter
def address(self, address):

    Sets the address of the wallet

    ==============================

    Parameters

    ----------

    address: str

    The address to be set as a wallet address.

    Throws

    ------

    PermissionError

    If you try to set the address more than once.

    PermissionError

    If the sate of the wallet doesn't cont WALLET_EDITABLE turned
    on.
```

```
@property
def details(self):

    Gets all details of the wallet

    ==============================

    Returns

    -------

    dict

    Dict of details of the wallet.


@property
def displayed_name(self):

    Gets the displayed name of the wallet

    =====================================

    Returns

    -------

    str

    The name that belongs to the wallet.
```

```
@displayed_name.setter
def displayed_name(self, name):

    Sets the displayed name of the wallet
    =====================================
    Parameters
    ----------
    name: str
    The name that belongs to the wallet.

    Throws
    ------
    PermissionError

    If the Wallet doesn't have WALLET_EDITABLE state.


def get_detail(self, key):

    Gets a detail of the wallet
    ===========================
    Parameters
    ----------
    key: str

    The name of the detail to get.

    Returns
    -------
    str

    The value of the given detail.

    None

    If the given key doesn't exist, the returned value is None.
```

```
def set_detail(self, key, value):

    Sets a detail of the wallet

    ===========================

    Parameters

    ----------

    key: str

    The name of the detail to set.

    value: str

    The new value of the given detail.

    Throws

    ------

    PermissionError

    If the Wallet doesn't have WALLET_EDITABLE state.


@property
def state(self):

    Gets the cummulated state of the wallet

    =======================================

    Returns

    -------

    int

    The cummulated state of the wallet.
```

Notes

State can consist of the presense or lack of the following constants:

WALLET_INSTANTIATED

WALLET_VALID

WALLET_EDITABLE

def state_add(self, stateid):

Adds the given state type to the wallet

=======================================

Parameters

----

stateid: int

State type to add to the state of the object.

Throws

ValueError

If the given state type is not a valid Wallet state type.

Notes

State types can be:

WALLET_INSTANTIATED

WALLET_VALID

WALLET_EDITABLE

```
def state_delete(self, stateid):

    Deletes the given state type from the wallet

    ==========================================

    Parameters

    ----------

    stateid: int

    State type to delete from the state of the object.

    Throws

    ------

    ValueError

    If the given state type is not a valid Wallet state type.

    Notes

    -----

    State types can be:

    WALLET_INSTANTIATED

    WALLET_VALID

    WALLET_EDITABLE
```

def state_toggle(self, stateid):

```
Toggles the given state type of the wallet

==========================================

Parameters

----------
```

stateid: int

```
State type to toggle in the state of the object.

Throws

------

ValueError

If the given state type is not a valid Wallet state type.

Notes

-----

State types can be:

WALLET_INSTANTIATED

WALLET_VALID

WALLET_EDITABLE
```

class UserWallet(Wallet):

This class represents a user account and wallet

def __init__(self, address, displayed_name=None, details=None, addressbook=None, documents=None, searches=None, transactions=None, utxos=None, unconfirmed_utxos=None):

```
Intializes the UserWallet object
================================
Parameters
----------
```

address: str

A bitcoincash address to register the user wallet to.

displayed_name str, optional (None if omitted)

The name of the user to be displayed.

details: dict, optional (None if omitted)

Data in key, value (str, str) format that represents details of the user wallet.

addressbook: WalletContainer, optional (None if omitted)

Container of foreign wallets that are registered by the user.

documents: list, optional (None if omitted)

Container of the document that are made by the user.

searches: list, optional (None if omitted)

Container of search activities of the user.

transaction: TransactionContainer, optional (None if omitted)

Container of all transactions of the user.

utxos: UtxoContainer, optional (None if omitted)

Container of all confirmed utxos of the user.

unconfirmed_utxos: UtxoContainer, optional (None if omitted)

Container of all unconfirmed utxos of the user.

Attributes

----

addressbook

documents

documents

searches

transactions

utxos

unconfirmed_utxos

Classmethods

----

from_address

from_file

from_dict_

Notes

1.

In contrast with common routines the creation of the UserWallet object through the usual init process is not suggested since it is not practical at all. Classmethods .from_address() and .from_file() provides much more convenient way to do this.

2.

However both .from_address() and .from_file() are actually wrappers of .from_dict_() classmethod it is strongly unadvised to use it directly. On the other hand for those who understand the workflow of the method it can be a good entry point to implement other ways to construct/restore an UserWallet instance.

@property

def addressbook(self):

Gets the addressbook of the user

================================

Returns

WalletContainer

The addressbook of the user.

Notes

However this property does not have setter, it doesn't mean that

the addressbook itself is not editable. It means only that it is

not replaceable.

```
@property
def documents(self):

    Gets the list of the documents created by the user

    ==================================================

    Returns

    -------

    list

    The list of the documents of the user.

    Notes

    -----

    However this property does not have setter, it doesn't mean
    that

    the list of documents itself is not editable. It means only
    thet

    it is not replaceable.


@classmethod
def from_address(cls, address):

    Creates a new UserWallet instance from bitcoincash address

    ==========================================================

    Parameters

    ----------

    address: str

    A bitcoincash address to create a new UserWallet from.
    Returns

    -------

    UserWallet

    The new instance for the user.

    Throws

    ------

    ValueError

    If the address is not valid.

    Notes

    -----

    1

    This is a classmethod and this method is a quite common and
    convenient way to create a user wallet especially if you use
    this code like part of online service and you register a new
    user.

    2

    The service works with legacy and SLP addresses as well but
    this code prefers to use bitcoincash address where possible.


@classmethod
def from_file(cls, filename):

    Creates a new UserWallet instance from data from file

    =====================================================

    Parameters
    ----------

    filename: str

    The name or path of a file to create a new UserWallet from.

    Returns

    -------

    UserWallet

    The new instance for the user.

    Throws

    ------

    FileNotFoundError

    If the file does not exist.

    Notes

    -----

    This is a classmethod and this method is a quite common and
    convenient way to create a user wallet especially if you use
    this code like part of any service if you want to restore a
    user from a saved state.


@property
def searches(self):

    Gets the list of the search experiments created by the user

    ============================================================

    Returns

    -------

    list
```

The list of the search experiments of the user.

Notes

However this property does not have setter, it doesn't mean that the list of searches itself is not editable. It means only that it is not replaceable.

```
@property
def transactions(self):
```

Gets the transactions of the user

=================================

Returns

TransactionContainer

The transactions of the user.

Notes

1.

However this property does not have setter, it doesn't mean that the transaction itself is not editable. It means only that it is not replaceable.

2.

It is strongly unadvised to delete from containers based on blockchain data. It is much better to re-query and re-generate the concerning object if something bad or unexpected happened.

```
@property
def utxos(self):

    Gets the utxos of the user

    ================================

    Returns

    -------

    UtxoContainer

    The utxos of the user.

    Notes

    -----

    1.

    However this property does not have setter, it doesn't mean
    that the utxos itself is not editable. It means only that it
    is not replaceable.

    2.

    It is strongly unadvised to delete from containers based on
    blockchain data. It is much better to re-query and re-generate
    the concerning object if something bad or unexpected happened.
```

```
@property
def unconfirmed_utxos(self):

    Gets the unconfirmed utxos of the user

    ======================================

    Returns

    -------

    UtxoContainer

    The unconfirmed utxos of the user.

    Notes

    -----

    1.

    However this property does not have setter, it doesn't mean
    that the unconfirmed utxos itself is not editable. It means
    only that it is not replaceable.

    2.

    It is strongly unadvised to delete from containers based on
    blockchain data. It is much better to re-query and re-generate
    the concerning object if something bad or unexpected happened.
```

```
@classmethod
def from_dict_(cls, datadict):

    Creates a new UserWallet instance from a dict

    =============================================

    Parameters

    ----------

    datadict: dict

    All the data which is needed to restore an existing UserWallet

    instance.

    Returns

    -------

    UserWallet

    The new instance for the user.

    Notes

    -----

    However this is a publicly accessible classmethod it is
    unadvised to use it directly for getting new UserWallet
    instances. On the other hand this method can be useful to
    restore UserWallet instances from different sources as well.
```

IV. About us

Axel Ország-Krisz Dr.

https://hyperrixel.com/en/axel

Richárd Ádám Vécsey Dr.

https://hyperrixel.com/en/richard

We are two deep learning developers and data scientists. We work in a lot of different fields of AI developing. Nowadays, most companies like to hire developers who are outsiders, who have other skills and education. We are lawyers who have jumped from law to programming and deep learning. We had a lot of opportunities from Facebook and Udacity. These companies maintain scholarship programs to get knowledge from people around the world. We were two of them. Computer Vision Nanodegree or Deep Learning Nanodegree were supported by Facebook. Besides the knowledge, we got a lot of new viewpoints and the possibility to help others. We are thankful and we are trying to give back something to the community and to the people around the Globe. Under these signs, we created a lot of open source projects.

We like to work in hackathon-style. Hackathon is more than a normal challenge where a bunch of people develop and everybody is a rival. It is an opportunity to get together, to know each other's way of thinking and to create connections that exist longer than a 2 or 3 days challenge. It is about to dare dream something and build from scratch within days. Hackathon is an attitude. Hackathon is a life-form. Every day is a hackathon in our life.

ChainBridge