Blockchains integration

Version: 1.0.0-rc-1
Update date: 2018-01-26

This document might be updated during the ongoing competition according to developers feedback. If you want to comment on this document please contact @olegzhelenkov or @KonstantinRyazantsev in the Lykke Dev telegram chat.

Versions history

Version	Date	Changes
0.3	2017-12-22	Wallet id removed
0.4	2017-12-25	Introduction is added
0.5	2017-12-25	Timestamp format in the Webhook changed from the unix timestamp to the ISO 8601
0.6	2017-12-29	Webhook is removed. Blockchain.Api contract is enhanced
0.7	2017-12-29	Blockchain.Api: - OperationId is made Guid - address field is added to the pending cashin event response - fromAddres field is added to the pending cashout-started, cashout-completed and cashout-failed events
0.8	2018-01-09	- Blockchain.Api is completely reworked - Diagrams are added - Description is added
0.9	2018-01-09	- In the in-progress, completed and failed transactions the “fee” field is added
0.10	2018-01-10	- balance field in the [get] /api/balances is replaced with inputTransactions field - description of the [get] /api/transactions/in-progress is updated - description of the [delete] /api/transactions/observations is updated - isDebug field name is fixed in the [get] /api/isalive
0.11	2018-01-10	- replacement of the balance field to the inputTransactions field is reverted (int the [get] /api/balances)
0.12	2018-01-10	- /api/transactions/history enpoints are added to the blockchain API - error response 406 Not acceptable is added to the [post] /api/transactions and to the [put] /api/transactions
0.13	2018-01-10	- “skip” is removed from the [get] /api/balances and [get] /api/transactions/xxx endpoints and replaced with continuation token. Responses are updated as well. - [get] /api/transactions/history endpoints are updated - [get] /api/assets endpoint is updated. Continuation based pagination is added
0.14	2018-01-10	- excess fields are removed from the [put] /api/transactions body
0.15	2018-01-13	- [get] /api/transactions/in-progress, [get]

		/api/transactions/completed, [get] /api/transactions/failed are replaced with the single endpoint [get] /api/transactions/broadcast/{operationId} - [delete] /api/transactions/observation is replaced with the [delete] /api/transactions/broadcast/{operationId}
0.16	2018-01-16	- amount and fee fields are made required only for the completed transactions in the [get] /api/transactions/broadcast/{operationId}
0.17	2018-01-16	- response error 501 is added for the [put] /api/transactions
1.0.0-rc-1	2018-01-26	- [get] /api/capabilities endpoint is added. - [get] /api/balances response is updated. block field is added - [post] /api/transactions/ is renamed to the /api/transactions/single, description and response are updated, HTTP status code errors are removed. errorCode field is added to the response. - optional endpoints [post] /api/transactions/many-inputs and [post] /api/transactions/many-outputs are added. - [put] /api/transactions description and response are updated. errorCode field is added to the response. - [post] /api/transactions/broadcast description is updated, response is added - [get] /api/transactions/broadcast/{operationId} is renamed to the [get] /api/transactions/broadcast/single/{operationId}, description and response are updated. errorCode and block fields are added to the response. - optional endpoints [get] /api/transactions/many-inputs and [get] /api/transactions/many-outputs are added. - [delete] /api/transactions/history/from/{address}/observation endpoint is added - [delete] /api/transactions/history/to/{address}/observation endpoint is added

Contents

Introduction

1. What this document is about

This document describes the contract between the Lykke Wallet platform and particular blockchain which someone wants to integrate into the Lykke Wallet platform. This integration is named as Blockchain Integration Layer.

2. To whom this document is addressed

This document may concern blockchain developers which want to integrate their blockchain into the Lykke Wallet platform.

3. Common requirements

Services, that implement the blockchain-specific part of the contract which is described in this document, should satisfy non-functional constraints listed in this section.

a. Technologies

- Source code should be written in one of the given languages: C# (.netcore), Java, C/C++, js (nodejs), Python
- If C# is used, services should be based on the template
- Azure Storage should be used as the persistent storage, if needed
- Mongo DB should be used as the indexing storage, if many indexes are needed
- Redis should be used as the cache, if needed

b. Design

- App should contain dedicated repositøries layer, in which all interaction with Azure Storage, Mongo DB and Redis should be encapsulated. Rest of the app should depends only on abstractions of these repositories (DI container configuration is exception).
- All atomic idempotent operations with external services (storage, cache, web-services) should be retried in case of transient failures (HTTP status code 400, 409, 412 are not transient failures for example)

c. Settings

- The app should read settings from the URL or local file specified in the SettingsUrl environment variable
    - What exactly (URL or path to the local file) is specified in the SettingsUrl should be determined automatically
- Settings will be provided in the Json format

- Settings should contain NetworkType field in the root element, if this is applicable for the given blockchain. Available values for this field are: Test, Main.
- At least, logs and data connection string should be specified in the different settings fields. More grained separation is at your discretion.
- When HTTP status code 403 is received as response of an request to the Azure Storage, performing by the app, settings should be reloaded and the request should be repeated
- If the app is implemented using C#, Lykke.SettingsReader Nuget package should be used to read the settings

d. Logs

- At least Info, Warning, Error and Monitor logging levels should be distinguished in the App
- The app should log all unhandled failures to the Error logging level
- All failed requests to the storage or cache should be logged to the Error logging level, with the entity as the context, only after all retries are done, if applicable.
- The app should log when it is started or stopped to the Monitor logging level with the messages "Started" and "Terminating" respectively.
- The app should write all log entries to the stdout in the format:
    - "{time:yyyy-MM-dd HH:mm:ss:fff} [{level}] {action description}:{context} - {message}"
- The app should write all log entries to the Azure Table storage in the format:
    - DateTime: DateTime - moment when the entry is logged
    - Level: string: {info, warning, error, monitor} - log level of the entry
    - Env: string - ENV_INFO environment variable value
    - Version: string - version of the app
    - Component: string - component, where the entry was logged (usually - class name)
    - Process: string - process, in which entry was logged (usually - method name)
    - Context: string - any usable data
    - Type: string - type of the exception, if any
    - Stack: string - call stack of the exception, if any
    - Msg: string - a message
- The app should log Warning, Error and Monitor levels to the Azure Storage Queue
    - Queue name should be taken from the SlackNotifications.AzureQueue.QueueName settings field
    - Connection string to the Azure Storage Queue should be taken from the SlackNotifications.AzureQueue.ConnectionString settings field
    - Message should be serialized as the Json with the given structure:

```
{
  // for Error, Warning, Monitor log levels respectively
  "type": "{Errors|Warning|Monitor}",
  // for Error, Warning, Monitor log levels respectively
  "sender": "{:exclamation:|:warning:|:information_source:}",
  "message": "a message"
}
```

- If the app is implemented using C#, Lykke.Common, Lykke.Logs and Lykke.SlackNotification.AzureQueue Nuget packages should be used to write the logs.

e. Deployment

- Step by step deployment guide should be provided.

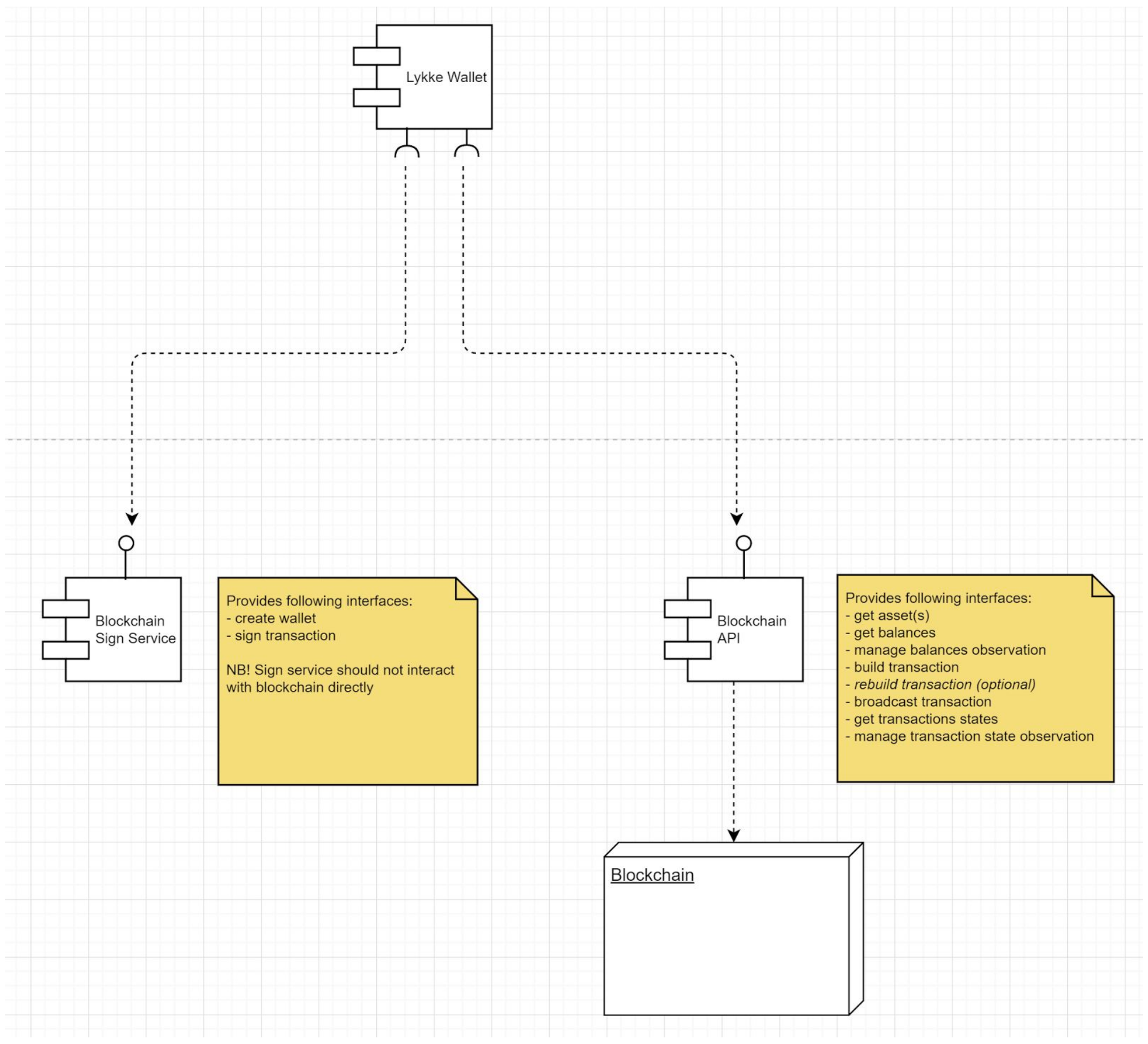
Abstract

Blockchain Integration Layer (BIL) affects the next modules:

- Common
    - Lykke Wallet - the platform into which blockchains are integrated
- Blockchain specific (should be implemented for the particular blockchain)
    - Blockchain.SignService - creates the wallets and implements transaction signing
    - Blockchain.Api - implements a unified adapter to the blockchain node

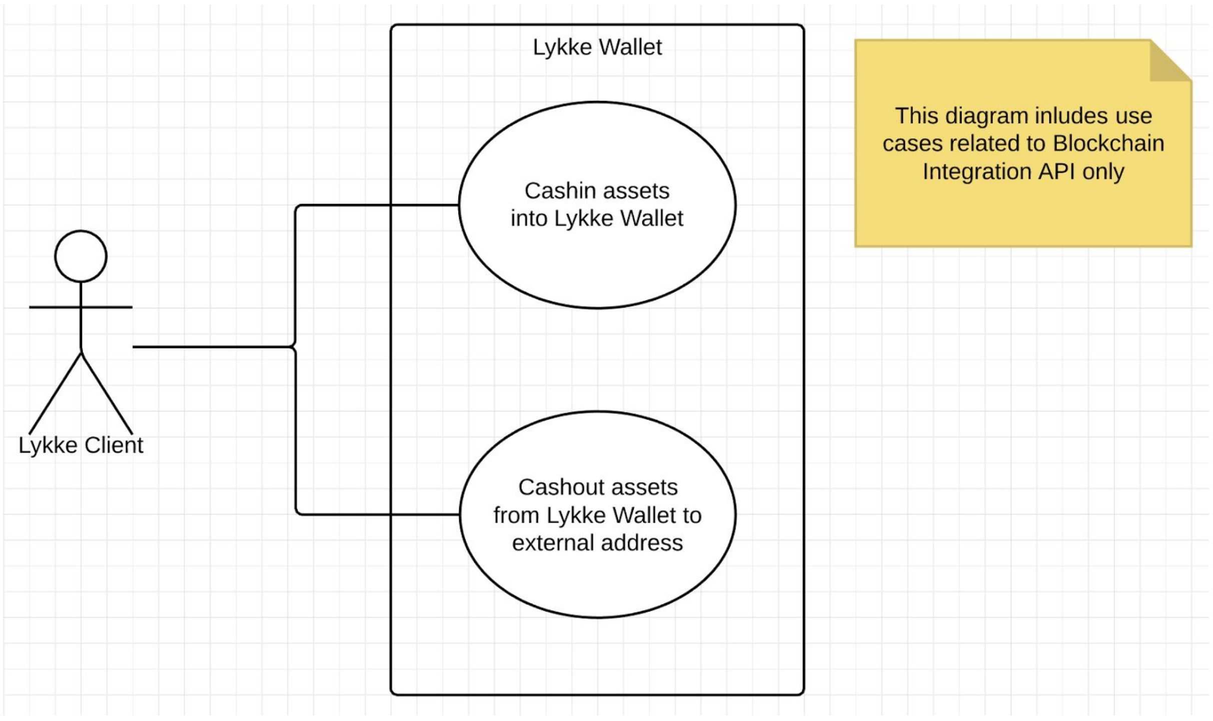
Diagrams cited in this article show architecture of the BIL.

Components

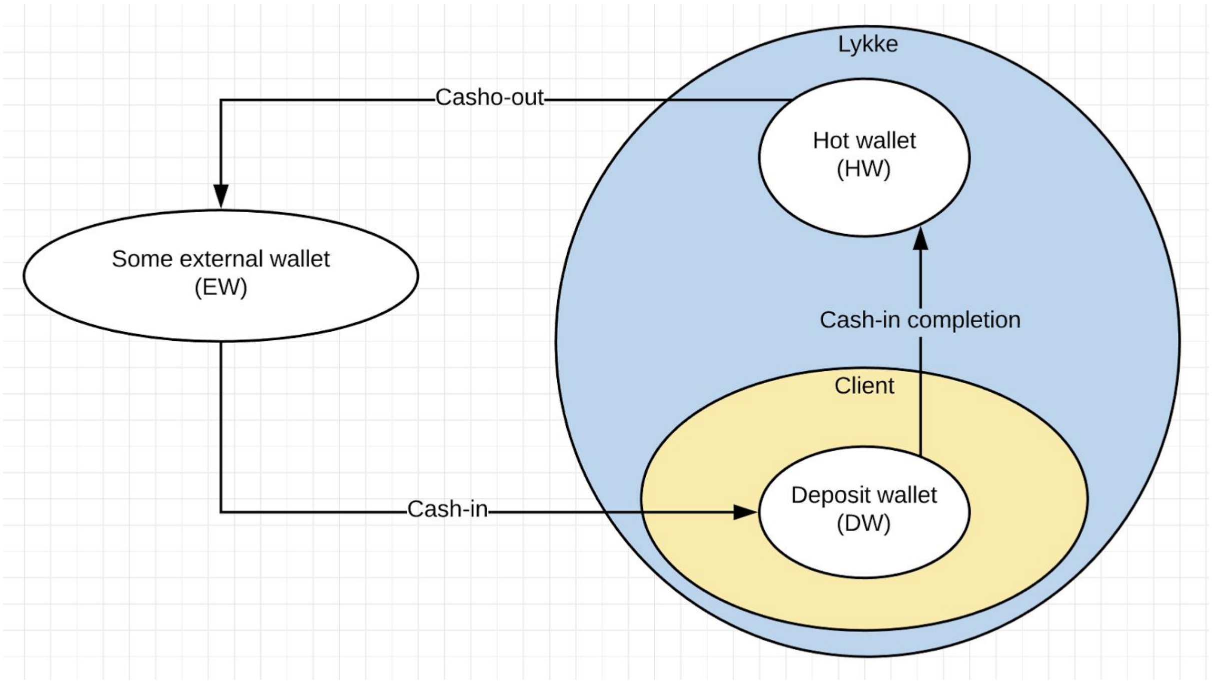


Lykke Wallet
Blockchain Sign Service
Provides following interfaces:
- create wallet
- sign transaction
NB! Sign service should not interact with blockchain directly
Blockchain API
Provides following interfaces:
- get asset(s)
- get balances
- manage balances observation
- build transaction
- *rebuild transaction (optional)*
- broadcast transaction
- get transactions states
- manage transaction state observation
Blockchain

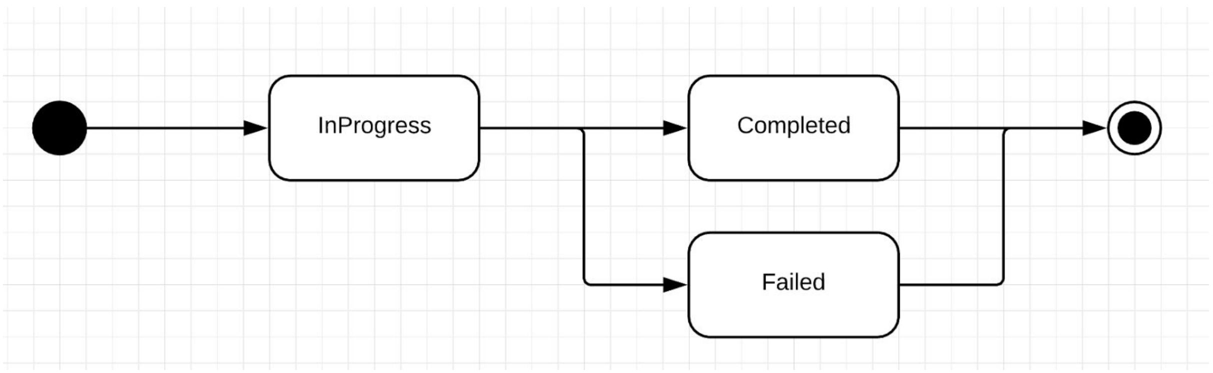
Lykke client use cases


Lykke Wallet
Cashin assets into Lykke Wallet
Cashout assets from Lykke Wallet to external address
Lykke Client
This diagram inludes use cases related to Blockchain Integration API only

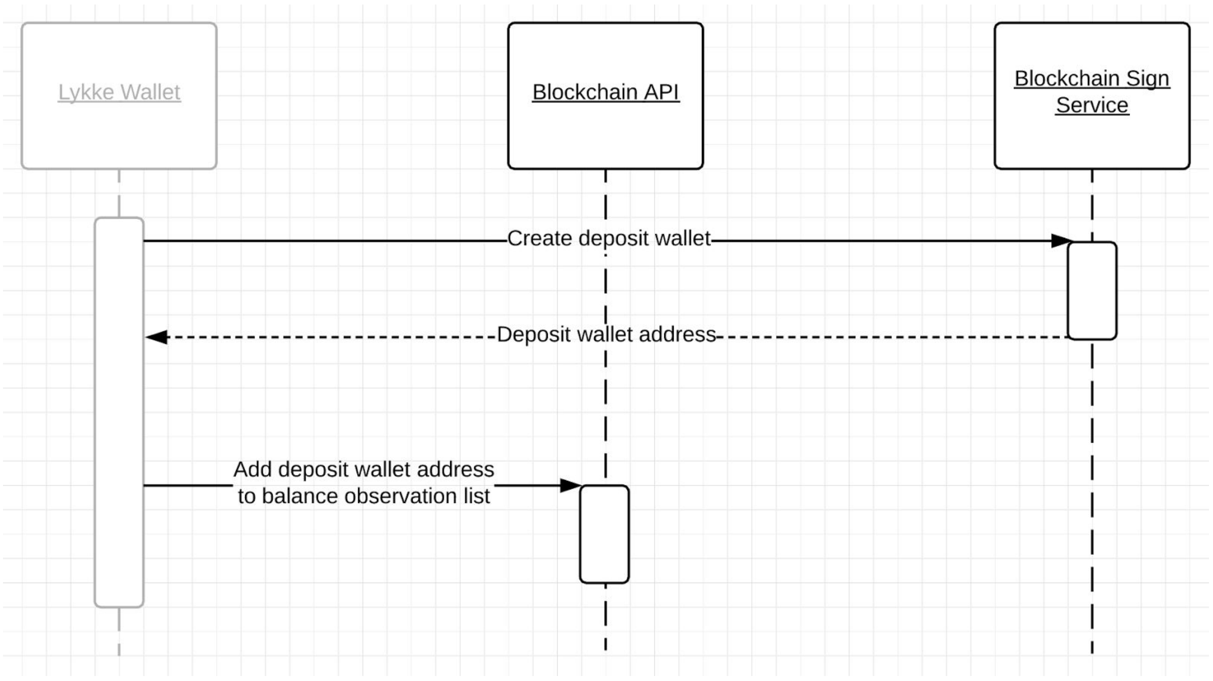
Fund flow



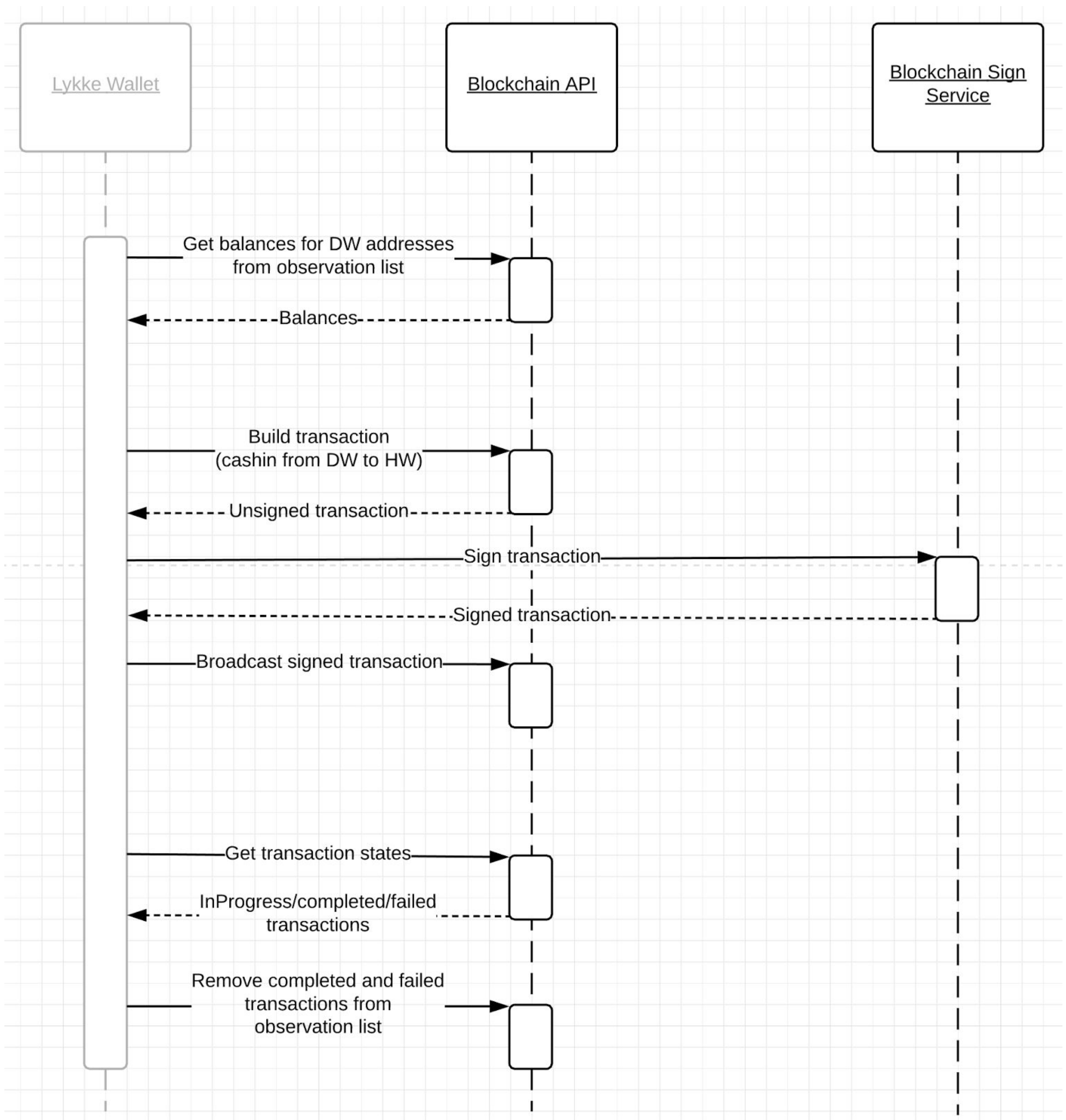
Transactions state transitions



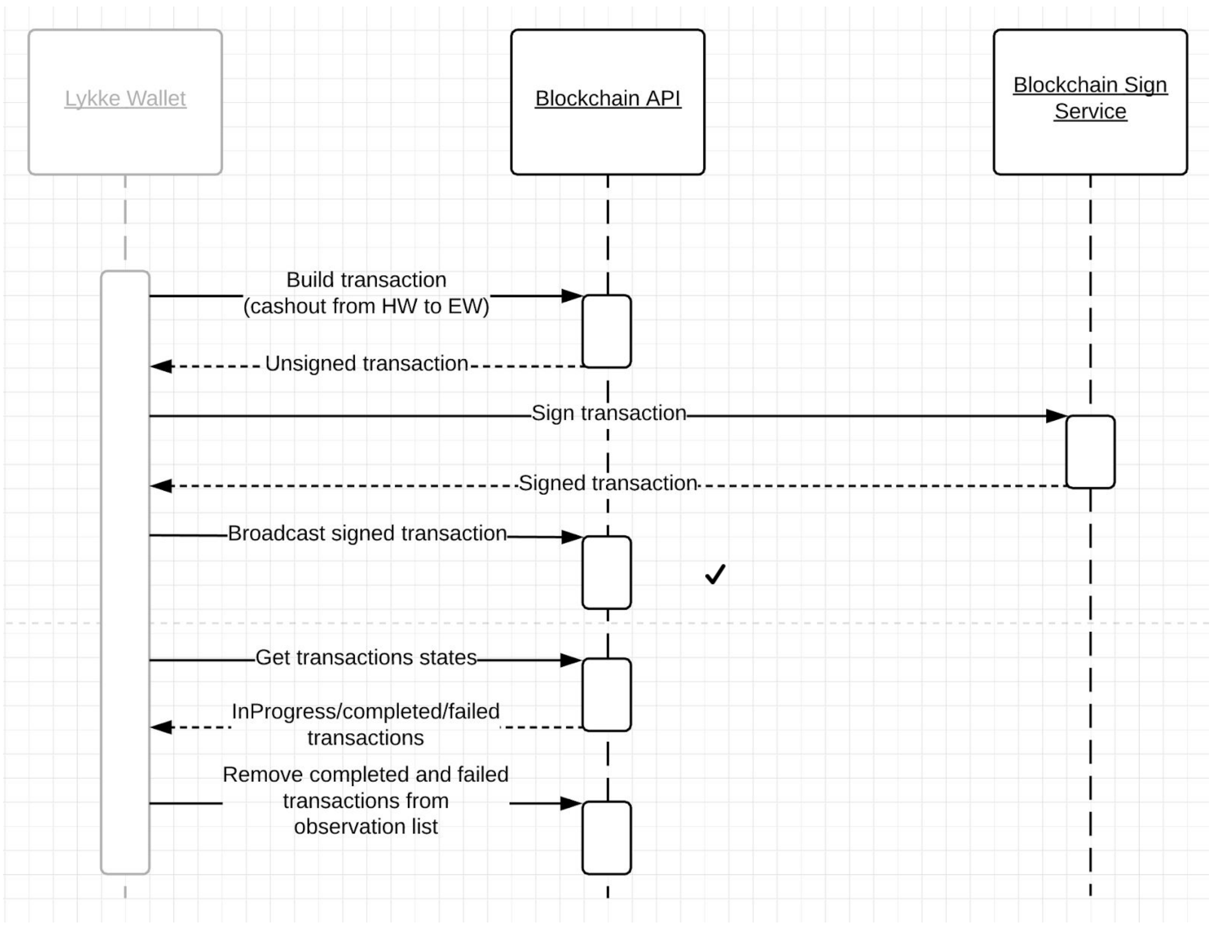
Wallet creation process



Cashin process


Lykke Wallet
Blockchain API
Blockchain Sign Service
Get balances for DW addresses from observation list
Balances
Build transaction (cashin from DW to HW)
Unsigned transaction
Sign transaction
Signed transaction
Broadcast signed transaction
Get transaction states
InProgress/completed/failed transactions
Remove completed and failed transactions from observation list

Cashout process


Lykke Wallet
Blockchain API
Blockchain Sign Service
Build transaction (cashout from HW to EW)
Unsigned transaction
Sign transaction
Signed transaction
Broadcast signed transaction
✓
Get transactions states
InProgress/completed/failed transactions
Remove completed and failed transactions from observation list

Contract

4. Common

All APIs described below should follow given rules unless otherwise is specified in the particular API.

Interface

HTTP REST API

General

- In case of an error with 4XX and 5XX HTTP status codes, API should return response with the following contract:

```
{
    // General error message, or summary
    // Can be empty
    “errorMessage”: “string”,

    // Errors related to the input model
    // Can be empty
    “modelErrors”: {

        // Collection of errors, related to the
        // particular field of the input model
        “<field-name>”: [
            “string”
        ]
    }
}
```

- APIs should use next HTTP status codes for the responses:
    - 200 Ok - request is completed successfully.
    - 400 Bad Request - invalid input data.
    - 500 Internal Server Error - unknown server error.

Methods

[GET] /api/isalive

Should return some general service info. Used to check if the service is running.

Response:

```
{
    // Name of the service
    “name”: “string”,
    // Version of the service
    “version”: “string”,
    // ENV_INFO environment variable value
    “env”: “string”,
    // Flag, which indicates if the service is built
    // in the debug configuration or not
    “isDebug”: boolean
}
```

5. Blockchain.SignService

Abstract

- This service should implement the abstract interface for the wallets (address) creation and transaction signing for the specific blockchain.
- The service shouldn’t store any sensitive data (addresses or private keys).
- The service shouldn’t depend on the blockchain node.
- The service will be used (called) by the Lykke platform to provide blockchain integration.

Methods

[POST] /api/wallets

Should create a new wallet (address) in the blockchain. This is a deposit wallet and it will be used as the client personal transit address to detect client cashin. This address will be displayed to the user in the app as the QR-code. It should be able to transfer funds to this address. Just after cashin detection, fund will be transferred to the hot wallet.

Response:

```
{
    // Private key, which will be used by Lykke platform to
    // sign transactions by the [POST] /api/sign
    “privateKey”: ”string”,

    // Address which identifies the wallet in the blockchain
    “publicAddress”: “string”
}
```

[POST] /api/sign

Should sign given transaction with the given private key

Body:
```
{
    // Private keys, which were returned by the
    // [POST] /api/wallets. Multiple keys can be used
    // for transactions with multiple inputs
    "privateKeys": [
        "string"
    ],

    // The transaction context in the blockchain
    // specific format, which was returned by the
    // Blockchain.Api
    // [POST] /api/transactions or [PUT] /api/transactions
    "transactionContext": "string"
 }
```

Response:
```
{
    // Signed transaction, which will be used to broadcast
    // the transaction by the Blockchain.Api
    // [PUT] /api/transactions/broadcast
    “signedTransaction”: “string”
}
```

6. Blockchain.API

Abstract

- This service should implement the abstract interface for the transactions management, balance monitoring and utility functions for the specific blockchain.
- The service should use the blockchain node to implement given interface.
- The service can store any intermediate data in the persistent storage or in the cache to provide functional and non-functional requirements.
- The service will be used (called) by the Lykke platform to provide blockchain integration.

Methods

[GET] /api/capabilities

Should return API capabilities set. Each optional operation has corresponding flag in the capabilities. Optional operations should be implemented if particular blockchain provides such functionality.

Response:

```
{
    // Should be true, if [PUT] /api/transactions call
    // is supported
    “isTransactionsRebuildingSupported”: boolean,

    // Should be true if
    // [POST] /api/transactions/many-inputs and
    // [GET] /api/transactions/broadcasted/many-inputs calls
    // are supported
    “areManyInputsSupported”: boolean,

    // Should be true if
    // [POST] /api/transactions/many-outputs and
    // [GET] /api/transactions/broadcasted/many-outputs calls
    // are supported
    “areManyOutputsSupported”: boolean
}
```

[GET] /api/assets?take=integer&[continuation=string]

Should return batch blockchain assets (coins, tags). Amount of the returned assets should not exceed take. Optional continuation contains context of the previous request, to let Blockchain.Api resume reading of the assets from the previous position. If continuation is empty, assets should be read from the beginning.

Response:

```
{
    // Continuation token, that
    // can be used to continue data reading
    // from the current position.
    // Should be null or empty string if no more data
    // to read.
    “continuation”: “string”,

    // Current batch of the items
    // Should be empty array if there are no items
    “items”:
    [
        {
            // Asset ID
            “assetId”: “string”,

            // Asset address, which identifies
            // asset in the blockchain, if applicable
            // for the given blockchain.
            // Can be empty
            “address”: “string”,

            // Asset display name
            “name”: “string”,

            // Asset accuracy - maximum number
            // of significant decimal digits to the right
            // of the decimal point in the asset amount.
            // Valid range: [0..28]
            “accuracy”: integer
        }
    ]
}
```

[GET] /api/assets/{assetId}

Should return specified asset (coin, tag)

Response:

```
{
    // Asset ID
    “assetId”: “string”,

    // Asset address, which identifies
    // asset in the blockchain, if applicable.
    // Can be empty
    “address”: “string”,

    // Asset display name
    “name”: “string”,

    // Asset accuracy - maximum number
    // of significant decimal digits to the right
    // of the decimal point in the asset amount.
    // Valid range: [0..28]
    “accuracy”: integer
}
```

Errors:

- 204 No content - specified asset not found

[GET] /api/addresses/{address}/validity

Should check and return wallet address validity

Response:

```
{
    // Flag, which indicates that the address is valid
    “isValid”: boolean
}
```

[POST] /api/balances/{address}/observation

Should remember the wallet address to observe the wallet balance and return it in the [GET] /api/balances, if the balance is non zero.

Errors:

- 409 Conflict - specified address is already observed.

[DELETE] /api/balances/{address}/observation

Should forget the wallet address and stop observing its balance.

Errors:

- 204 No content - specified address is not observed

[GET] /api/balances?take=integer&[continuation=string]

Should return balances of the observed wallets with non zero balances. Wallets balance observation is enabled by the [POST] /api/balances/{address}/observation and disabled by the [DELETE] /api/balances/{address}/observation.
Amount of the returned balances should not exceed take. Optional continuation contains context of the previous request, to let Blockchain.Api resume reading of the balances from the previous position. If continuation is empty, balances should be read from the beginning.

Response:

```
{
    // Continuation token, that
    // can be used to continue data reading
    // from the current position.
    // Should be null or empty string if no more data
    // to read.
    “continuation”: “string”,

    // Current batch of the items
    // Should be empty array if there are no items
    “items”:
    [
        {
            // Wallet address
            “address”: “string”,

            // Asset ID
            “assetId”: “string”,

            // Balance is integer as string, aligned
            // to the asset accuracy. Actual value can be
            // calculated as
            // x = sourceBalance * (10 ^ asset.Accuracy)
            “balance”: “string”,

            // Incremental ID of the moment, when balance
            // is updated. It should be the same sequence
            // as for block in the
            // [GET] /api/transactions/broadcast/* responses
            // For the most blockchains it could be the
            // block number/height.
            “block”: integer64
        }
    ]
}
```

[POST] /api/transactions/single

Should build not signed transaction to transfer from the single source to the single destination. If the transaction with the specified operationId has already been built by one of the [POST] /api/transactions/* call, it should be ignored and regular response (as in the first request) should be returned.

Body:

```
{
    // Lykke unique operation ID
    “operationId”: “guid”,
    // Source address
    “fromAddress”: “string”,

    // Destination address
    “toAddress”: “string”,

    // Asset ID to transfer
    “assetId”: “string”,

    // Amount to transfer. Integer as string, aligned
    // to the asset accuracy. Actual value can be
    // calculated as
    // x = amount / (10 ^ asset.Accuracy)
    “amount”: “string”,

    // Flag, which indicates, that the fee should be included
    // in the specified amount.
    // Example:
    // if(includeFee == true) actualAmount = amount
    // if(includeFee == false) actualAmount = amount + fee
    “includeFee”: boolean
}
```

Response:

```
{
    // Error code.
    // Can be empty.
    // Should be non empty if an error that match one of the
    // listed code is occured. For other errors use HTTP
    // status codes.
    // enum values:
    // - amountIsTooSmall: amount is too small to execute
    // transaction
    // - notEnoughBalance: transaction can’t be executed due
    // to balance insufficiency on the source address.
    "errorCode": "enum",

    // The transaction context in the blockchain
    // specific format, which will be passed to the
    // Blockchain.SignService [POST] /api/sign.
    // Should be not empty when result is successful.
    “transactionContext”: “string”
}
```

[POST] /api/transactions/many-inputs

Optional

See [GET] /api/capabilities

Should build not signed transaction with many inputs. If the transaction with the specified operationId has already been built by one of the [POST] /api/transactions/* call, it should be ignored and regular response (as in the first request) should be returned. Fee should be included in the specified amount.

Body:

```
{
    // Lykke unique operation ID
    “operationId”: “guid”,

    // Sources
    “inputs”: [
        {
            // Source address
            “fromAddress”: “string”,

            // Amount to transfer from the fromAddress.
            // Integer as string, aligned
            // to the asset accuracy. Actual value can be
            // calculated as
            // x = amount / (10 ^ asset.Accuracy)
            “amount”: “string”
        }
    ],

    // Destination address
    “toAddress”: “string”,

    // Asset ID to transfer
    “assetId”: “string”
}
```

Response:

```
{
    // Error code.
    // Can be empty.
    // Should be non empty if an error that match one of the
    // listed code is occured. For other errors use HTTP
    // status codes.
    // enum values:
    // - amountIsTooSmall: amount is too small to execute
    // transaction
    // - notEnoughBalance: transaction can't be executed due
    // to balance insufficiency on the source address.
    "errorCode": "enum",

    // The transaction context in the blockchain
    // specific format, which will be passed to the
    // Blockchain.SignService [POST] /api/sign
    // Should be not empty when result is successful.
    "transactionContext": "string"
}
```

Errors:

- 501 Not Implemented - function is not implemented in the blockchain.

[POST] /api/transactions/many-outputs

Optional
See [GET] /api/capabilities

Should build not signed transaction with many outputs. If the transaction with the specified operationId has already been built by one of the [POST] /api/transactions/* call, it should be ignored and regular response (as in the first request) should be returned. Fee should be added to the specified amount.

Body:

```
{
    // Lykke unique operation ID
    "operationId": "guid",

    // Source address
    "fromAddress": "string",

    // Destinations
    "outputs": [
        {
            // Destination address
            "toAddress": "string",

            // Amount to transfer to the toAddress.
            // Integer as string, aligned
            // to the asset accuracy. Actual value can be
            // calculated as
            // x = amount / (10 ^ asset.Accuracy)
            "amount": "string"
        }
    ],

    // Asset ID to transfer
   “assetId”: “string”
}
```

Response:

```
{
    // Error code.
    // Can be empty.
    // Should be non empty if an error that match one of the
    // listed code is occured. For other errors use HTTP
    // status codes.
    // enum values:
    // - amountIsTooSmall: amount is too small to execute
    // transaction
    // - notEnoughBalance: transaction can’t be executed due
    // to balance insufficiency on the source address.
    "errorCode": "enum",

    // The transaction context in the blockchain
    // specific format, which will be passed to the
    // Blockchain.SignService [POST] /api/sign.
    // Should be not empty when result is successful.
    “transactionContext”: “string”
}
```

Errors:

- 501 Not Implemented - function is not implemented in the blockchain.

[PUT] /api/transactions

Optional
See [GET] /api/capabilities

Should rebuild not signed transaction with the specified fee factor, if applicable for the given blockchain. This should be implemented, if blockchain allows transaction rebuilding (substitution) with a new fee. This will be called if transaction is stuck in the “in-progress” state for too long, to try to execute transaction with a higher fee. [POST] /api/transactions/* with the same operationId should precede to the given call. Transaction should be rebuilt with parameters that were passed to the [POST] /api/transactions/*.

Body:

```
{
    // Lykke unique operation ID
    “operationId”: “guid”,

    // Multiplier for the original transaction fee.
    // Blockchain should multiply a regular fee
    // by this factor.
    // x = feeFactor * regularFee
    “feeFactor”: float
}
```

Response:

```
{
    // Error code.
    // Can be empty.
    // Should be non empty if an error that match one of the
    // listed code is occured. For other errors use HTTP
    // status codes.
    // enum values:
    // - amountIsTooSmall: amount is too small to execute
    // transaction
    // - notEnoughBalance: transaction can’t be executed due
    // to balance insufficiency on the source address.
    "errorCode": "enum",

    // The transaction context in the blockchain
    // specific format, which will be passed to the
    // Blockchain.SignService [POST] /api/sign
    // Should be not empty when result is successful.
    “transactionContext”: “string”
}
```

Errors:

- 501 Not Implemented - function is not implemented in the blockchain.

[POST] /api/transactions/broadcast

Should broadcast the signed transaction. errorCode can be evaluated synchronously or returned asynchronously in the [GET] /api/transactions/broadcast/* response with errorCode, depending on the particular blockchain implementation

Body:

```
{
    // Lykke unique operation ID
    “operationId”: “guid”,

    // The signed transaction returned by the
    // Blockchain.SignService [POST] /api/sign
    “signedTransaction”: “string”
}
```

Response:

```
{
    // Error code.
    // Can be empty.
    // Should be non empty if an error that match one of the
    // listed code is occured. For other errors use HTTP
    // status codes.
    // enum values:
    // - amountIsTooSmall: amount is too small to execute
    // transaction
    // - notEnoughBalance: transaction can’t be executed due
    // to balance insufficiency on the source address.
    "errorCode": "enum",
}
```

Errors:

- 409 Conflict - transaction with specified operationId and signedTransaction has already been broadcasted.

[GET] /api/transactions/broadcast/single/{operationId}

Should return broadcasted transaction by the operationId. All transactions with single input and output, that were broadcasted by the [POST] /api/transactions/broadcast should be available here.

Response:

```
{
    // Lykke unique operation ID.
    “operationId”: “guid”,

    // State of the transaction
    // enum values:
    // - inProgress: transaction is being in-progress
    // - completed: transaction is completed for sure
    // - failed: transaction is failed, if applicable for the
    //     particular blockchain
    “state”: “enum”,

    // Transaction moment as ISO 8601 in UTC
    “timestamp”: “datetime”,

    // Amount without fee.
    // Is integer as string, aligned to the asset accuracy.
    // Actual value can be calculated as
    // x = sourceAmount * (10 ^ asset.Accuracy)
    // Should be non empty if the state is Completed
    “amount”: “string”,

    // Fee. Is integer as string, aligned
    // to the asset accuracy. Actual value can be
    // calculated as
    // x = sourceFee * (10 ^ asset.Accuracy)
    // Should be non empty if the state is Completed
    “fee”: “string”,

    // Transaction hash as base64 string.
    // Can be empty
    // Should be non empty if the state is Completed
    “hash”: “string”,

    // Error description
    // Can be empty
    // Should be non empty if the state is Error
    “error”: “string”,

    // Error code.
    // Can be empty.
    // Should be non empty if the state is Failed.
    // enum values:
    // - unknown: any error that does not fit another codes.
    // - amountIsTooSmall: amount is too small to execute
    // transaction
    // - notEnoughBalance: transaction can’t be executed due
    // to balance insufficiency on the source address.
    // Should be non empty if the state is Error
    “errorCode”: “enum”,

    // Incremental ID of the moment, when the transaction
    // state changing is detected. It should be the same
    // sequence as for block in the [GET] /api/balances
    // response. For the most blockchains it could be the
    // block number/height.
    “block”: integer64
}
```

Errors:

- 204 No content - specified transaction not found

[GET] /api/transactions/broadcast/many-inputs/{operationId}

Optional

See [GET] /api/capabilities

Should return broadcasted transaction by the operationId. All transactions with many inputs, that were broadcasted by the [POST] /api/transactions/broadcast should be available here.

Response:

```
{
    // Lykke unique operation ID.
    “operationId”: “guid”,

    // State of the transaction
    // enum values:
    // - inProgress: transaction is being in-progress
    // - completed: transaction is completed for sure. When
    //    transaction is switched to the completed state,
    //    amount of this transaction should be already
    //    subtracted from the balance returned by the
    //    [GET] /api/balances, for outgoing transactions
    // - failed: transaction is failed, if applicable for the
    //     particular blockchain
    “state”: “enum”,

    // Transaction moment as ISO 8601 in UTC
    “timestamp”: “datetime”,

    // Sources.
    // Should be non null if the state is Completed.
    “inputs”: [
        {
            // From address
            “fromAddress”: “string”,

            // Actual amount, that is transferred from the
            // fromAddress. Integer as string, aligned
            // to the asset accuracy. Actual value can be
            // calculated as
            // x = amount / (10 ^ asset.Accuracy)
            “amount”: “string”
        }
    ],
    // Destination address
    “toAddress”: “string”,

    // Fee. Is integer as string, aligned
    // to the asset accuracy. Actual value can be
    // calculated as
    // x = sourceFee * (10 ^ asset.Accuracy)
    // Should be non empty if the state is Completed
    “fee”: “string”,

    // Transaction hash as base64 string.
    // Can be empty
    // Should be non empty if the state is Completed
    “hash”: “string”,

    // Error description
    // Can be empty
    // Should be non empty if the state is Error
    “error”: “string”,

    // Error code.
    // Can be empty.
    // Should be non empty if the state is Failed.
    // enum values:
    // - unknown: any error that does not fit another codes.
    // - amountIsTooSmall: amount is too small to execute
    // transaction
    // - notEnoughBalance: transaction can’t be executed due
    // to balance insufficiency on the source address.
    “errorCode”: “enum”,

    // Incremental ID of the moment, when the transaction
    // state changing is detected. It should be the same
    // sequence as for block in the [GET] /api/balances
    // response. For the most blockchains it could be the
    // block number/height.
    “block”: integer64
}
```

Errors:

- 501 Not Implemented - function is not implemented in the blockchain.
- 204 No content - specified transaction not found

[GET] /api/transactions/broadcast/many-outputs/{operationId}

Optional
See [GET] /api/capabilities

Should return broadcasted transaction by the operationId. All transactions with many outputs, that were broadcasted by the [POST] /api/transactions/broadcast should be available here.

Response:

```
{
    // Lykke unique operation ID.
    “operationId”: “guid”,

    // State of the transaction
    // enum values:
    // - inProgress: transaction is being in-progress
    // - completed: transaction is completed for sure. When
    //    transaction is switched to the completed state,
    //    amount of this transaction should be already
    //    subtracted from the balance returned by the
    //    [GET] /api/balances, for outgoing transactions
    // - failed: transaction is failed, if applicable for the
    //     particular blockchain
    “state”: “enum”,

    // Transaction moment as ISO 8601 in UTC
    “timestamp”: “datetime”,

    // Source address
    “fromAddress”: “string”,

    // Destinations.
    // Should be non null if the state is Completed.
    “outputs”: [
        {
            // Destination address
            “toAddress”: “string”,

            // Actual amount that is transferred to the
            // toAddress. Integer as string, aligned
            // to the asset accuracy. Actual value can be
            // calculated as
            // x = amount / (10 ^ asset.Accuracy)
            “amount”: “string”
        }
    ],
    // Fee. Is integer as string, aligned
    // to the asset accuracy. Actual value can be
    // calculated as
    // x = sourceFee * (10 ^ asset.Accuracy)
    // Should be non empty if the state is Completed
    “fee”: “string”,

    // Transaction hash as base64 string.
    // Can be empty
    // Should be non empty if the state is Completed
    “hash”: “string”,

    // Error description
    // Can be empty
    // Should be non empty if the state is Error
    “error”: “string”,

    // Error code.
    // Can be empty.
    // Should be non empty if the state is Failed.
    // enum values:
    // - unknown: any error that does not fit another codes.
    // - amountIsTooSmall: amount is too small to execute
    // transaction
    // - notEnoughBalance: transaction can’t be executed due
    // to balance insufficiency on the source address.
    // Should be non empty if the state is Error
    “errorCode”: “enum”,

    // Incremental ID of the moment, when the transaction
    // state changing is detected. It should be the same
    // sequence as for block in the [GET] /api/balances
    // response. For the most blockchains it could be the
    // block number/height.
    “block”: integer64
}
```

Errors:

- 501 Not Implemented - function is not implemented in the blockchain.
- 204 No content - specified transaction not found

[DELETE] /api/transactions/broadcast/{operationId}

Should remove specified transaction from the broadcasted transactions. Should affect transactions returned by the [GET] /api/transactions/broadcast/{operatioId}.

Errors:

- 204 No content - specified transaction not found

[POST] /api/transactions/history/from/{address}/observation

Should start observation of the transactions that transfer fund from the address. Should affect result of the [GET] /api/transactions/history/from/{address}.

Errors:

- 409 Conflict: transactions from the address are already observed.

[POST] /api/transactions/history/to/{address}/observation

Should start observation of the transactions that transfer fund to the address. Should affect result of the [GET] /api/transactions/history/to/{address}.

Errors:

- 409 Conflict: transactions to the address are already observed.

[GET] /api/transactions/history/from/{address}?take=integer&[afterHash=string]

Should return completed transactions that transfer fund from the address and that were broadcasted after the transaction with the hash equal to the afterHash.
If afterHash is empty, transactions should be read from the beginning.
Should include transactions broadcasted not using this API.
If there are no transactions to return, empty array should be returned.
Amount of the returned transactions should not exceed take.

Response:

```
[
    {
        // Lykke unique operation ID.
        // Can be empty.
        // Should be not empty for transactions that
        // broadcasted using this Blockchain.Api
        “operationId”: “guid”,

        // Transaction moment as ISO 8601 in UTC
        “timestamp”: “datetime”,
        // Source address
        “fromAddress”: “string”,

        // Destination address
        “toAddress”: “string”,

        // Asset ID
        “assetId”: “string”

        // Amount without fee. Is integer as string, aligned
        // to the asset accuracy. Actual value can be
        // calculated as
        // x = sourceAmount * (10 ^ asset.Accuracy)
        “amount”: “string”,

        // Transaction hash as base64 string
        “hash”: “string”
    }
]
```

[GET] /api/transactions/history/to/{address}?take=integer&[afterHash=string]

Should return completed transactions that transfer fund to the address and that were broadcasted after the transaction with the hash equal to the afterHash.
If afterHash is empty, transactions should be read from the beginning.
Should include transactions broadcasted not using this API.
If there are no transactions to return, empty array should be returned.
Amount of the returned transactions should not exceed take.

Response:

```
[
    {
        // Lykke unique operation ID.
        // Can be empty.
        // Should be not empty for transactions that
        // broadcasted using this Blockchain.Api
        “operationId”: “guid”,

        // Transaction moment as ISO 8601 in UTC
        “timestamp”: “datetime”,

        // Source address
        “fromAddress”: “string”,

        // Destination address
        “toAddress”: “string”,

        // Asset ID
        “assetId”: “string”

        // Amount without fee. Is integer as string, aligned
        // to the asset accuracy. Actual value can be
        // calculated as
        // x = sourceAmount * (10 ^ asset.Accuracy)
        “amount”: “string”,

        // Transaction hash as base64 string
        “hash”: “string”
    }
]
```

[DELETE] /api/transactions/history/from/{address}/observation

Should stop observation of the transactions that transfer fund from the address. Should affect result of the [GET] /api/transactions/history/from/{address}.

Errors:

- 204 No content: transactions from the address are not observed.

[DELETE] /api/transactions/history/to/{address}/observation

Should stop observation of the transactions that transfer fund to the address. Should affect result of the [GET] /api/transactions/history/to/{address}.

Errors:

- 204 No content: transactions to the address are not observed.

Examples

Coming soon

Questions:

1. Fees?
2. Encryption for sign API

3. Assets management
4. Support / test envs / blockchain lifecycle

DOC TODO:
Performance requirements
Automated testing + description