BIZONEX

API USE
ON BIZONEX.COM STOCK EXCHANGE

Revision 1.3

SYSTEM DESCRIPTION

2020

Annotation

This document describes the structure of API information input, processing, storage and deletion on BIZONEX Stock Exchange.

The document consists of two sections. The first section describes the process of Token creation, setup and deletion on the Stock Exchange page in "User information" section. The second section describes the request methods allowing to perform transactions with API-keys, and describes the errors occurring in methods execution.

List of Annexes:

1. Annex A "HTTP request for Stock Exchange API".
2. Annex B "Websocket request for Stock Exchange API".

The document contains the lists of abbreviations and symbols, terms and definitions.

This document includes section describing documents revisions up to the current version.

CONTENT

1. General 4

1 Bot-programs setup on the BIZONEX Stock Exchange 5

1.1 User information 5

1.2 Verification procedure 6

1.3 Functional purpose of API-keys (Token) 8

1.4 Token creation 8

1.5 Token disclosure 9

1.6 Token deletion 9

2 Description of queries 11

2.1 Queries structure 11

2.2 Methods description 15

2.3 Output data 34

2.4 Error description 35

2. Annex A. HTTP request for Stock Exchange API 38

3. Annex B. Websocket - request for Stock Exchange API 39

4. LIST OF ABBREVIATIONS AND SYMBOLS 40

5. LIST OF TERMS AND DEFINITIONS 41

6. Versions of the documents 42

1. General

Bizonex.com (hereinafter - the Stock Exchange) operates as an electronic application called from the browser window when going by address: https://bizonex.com/.

BIZONEX Stock Exchange is a cryptocurrency stock exchange allowing to trade between foreign currencies (USD, Euros, RUB) and cryptocurrency types based on purchase or sale orders.

The documents describe the processes of the Tokens creation, setup and deletion on the Stock Exchange page in "User information" section, and provide for the request methods allowing to perform transactions with the API-keys.

Requests format is "POST". Requests use public or private methods. To make a request for the private methods, one should have valid public key and corresponding secret key.

NOTE: a fee in amount of 0.2% is charged on the Stock Exchange from each transaction performed under purchase or sale orders.

1 Bot-programs setup on the BIZONEX Stock Exchange

Bot-programs (hereinafter - the bots) are designed to perform transactions on the cryptocurrency stock exchange on behalf of the user.

Stock Exchange bots for automatic cryptocurrency trading operate according to the special algorithms created based on the number of potential losses and profits analysis during the accounting period. Based on this analysis, the rules for the robot trading strategy on the cryptocurrency Stock Exchange shall be formulated.

Stock Exchange bots operating within the specified algorithm have a number of advantages over the trader in transactions:

- rapid decision-making in orders execution;
- ability to analyse an unlimited number of pairs;
- round-the-clock transactions.

When registering on the BIZONEX Stock Exchange, bot can be created (API-key, Token) on the "User information" page in "API-keys" section. A new API-key (Token) can be created in the section. The number of Token transactions available is limited by the array of settings provided by the user.

Stock Exchange bot created operates only on a single trading platform - bizonex.com, and bots are free for the user.

0.2% fee is charged only for the sale and purchase transactions performed.

1.1 User profile

Shifting to the page takes place, when placing the mouse cursor on the icon with the first letter of the user login in the top right corner of the screen of BIZONEX Exchange and selecting section “User profile” in the context menu (Figure 1):

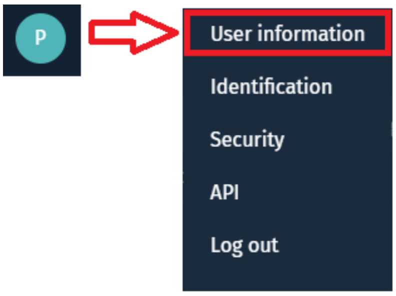


Figure 1 - Context menu with the list of settings available for the user

When moving along tab pages, the active tab is marked with turquoise colour in the navigation menu (Figure 2).

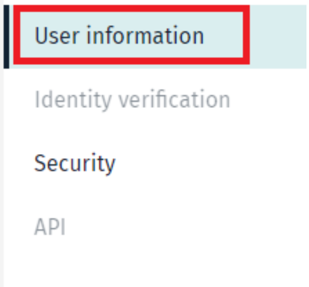


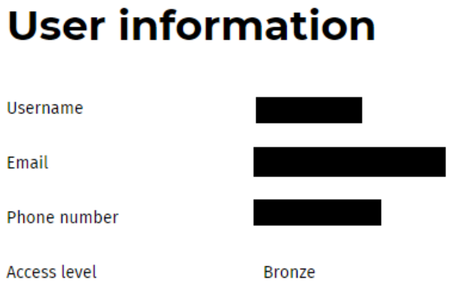
Figure 2 - Navigation menu and page “User profile”

List of the user settings:

1 “User profile” - general information about the user status (Figure 3):

- user name (login);
- e-mail address;
- phone number;

- access level (bronze, silver, gold, platinum) according to KYC-verification (for more details refer to “AML / KYC Policy” on the Exchange page).



User information

Username	[redacted]
Email	[redacted]
Phone number	[redacted]
Access level	Bronze

Figure 3 – Section “User profile” after completion of the registration procedure

The information in this section is not editable

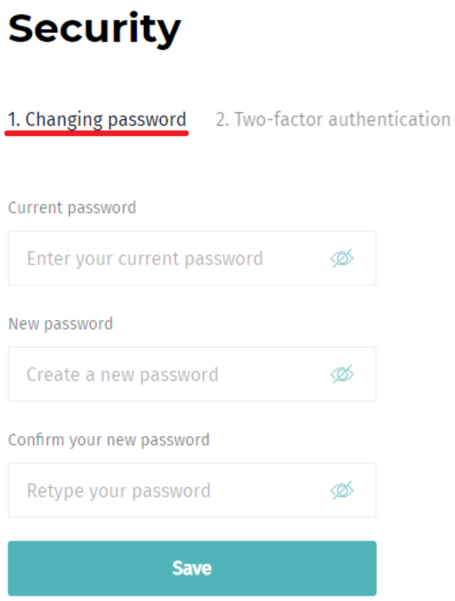
2 “Identification” / “Identity authentication” - personal data provided by the user for passing the verification procedure:

- Personal information.
- Address.
- Documents.
- Photos and consents.
- Income source.

After sending the data for checking the section is restricted for editing. When changing the data it is required to apply to the customer support service, having filled the application in the section “Support service > Send ticket”.

In case of providing the unauthentic or incomplete information the verification will not be passed.

3 “Security” - settings of the two-factor authentication and change of the user password.



Security

1. Changing password   2. Two-factor authentication

Current password

Enter your current password

New password

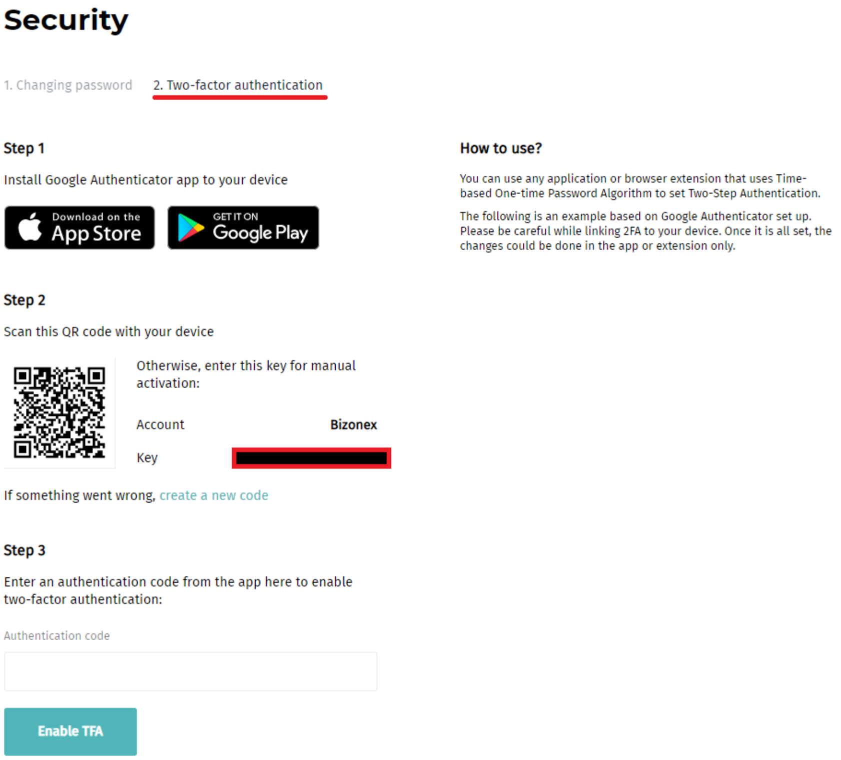
Create a new password

Confirm your new password

Retype your password

Save

Figure 4 – Change of the user password



Security

Step 1

Install Google Authenticator app to your device

How to use?

You can use any application or browser extension that uses Time-based One-time Password Algorithm to set Two-Step Authentication.

The following is an example based on Google Authenticator set up. Please be careful while linking 2FA to your device. Once it is all set, the changes could be done in the app or extension only.

Step 2

Scan this QR code with your device

Otherwise, enter this key for manual activation:

Account Bizonex

Key [redacted]

If something went wrong, create a new code

Step 3

Enter an authentication code from the app here to enable two-factor authentication:

Authentication code

Figure 5 – Setting of the two-factor authentication

4 “API” - form of “bot” creation, which perform operations on as the user.

5 “Exit” - button of exit of the user profile.

The document describes in detail only the sections “Identification” / “Identity authentication” and “API”. Description of the full functionality of BIZONEX Exchange is beyond the scope of the given document.

1.2 Completion of the user identification (verification) procedure

User identification (verification) procedure is the procedure for verification of authenticity of the provided personal data (personal information, address, income information) and scanned copies of the documents (documents, photos of the user).

After registration is completed the user is assigned with initial level of verification - “bronze” (filed “Access level”). The following levels (silver, gold, platinum) are accessible after filling the tabs of the section “Identification” / “Identity authentication” and confirmation that the provided information is authentic. In case the user is not verified according to the check results, then the letter containing the additional information will be sent to his e-mail address.

Table 1 - User access levels according to KYC-verification

No.	Access level	Functional capabilities of the user
1	**Bronze**	a) deposit in any crypto currency (tokens) is prohibited; b) withdrawal of funds in crypto currency (tokens) is prohibited; c) making of transactions is prohibited.
2	**Silver**	a) deposit in any crypto currency (tokens) is allowed; b) withdrawal of funds in crypto currency (tokens) with the limit equivalent to EUR 5000 (within 30 days) is allowed; c) making of transactions between any crypto currency (tokenized) pairs with the limit for trade amount equivalent to EUR 5000 (within 30 days) is allowed.
3	**Gold**	a) deposit in any crypto currency (tokens) is allowed;

		b) withdrawal of funds in crypto currency (tokens) with the limit equivalent to EUR 15,000 (within 30 days) is allowed; c) making of transactions between any crypto currency pairs with the limit for trade amount equivalent to EUR 50,000 (within 30 days) is allowed.
4	**Platinum**	a) deposit in any crypto currency (tokens) is allowed; b) withdrawal of any crypto currency (tokens) without restrictions; c) making of transactions between any two pairs without trade restrictions.

NOTES:

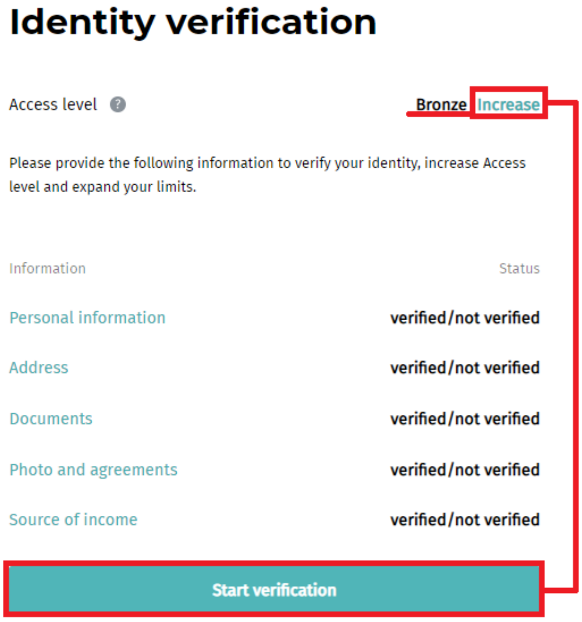
1 The conditions being provided to the users with respect to the access level can be revised at the trading platform of the Exchange, in case the Company's policy is changed.

2 Withdrawal of funds and making of transactions for particular users can be suspended irrespective of their current access level in case the false information have been provided:

- regarding personal data;
- regarding income source.

The user's funds in this case will be frozen in the moment of conducting the check and until the legitimacy of making the operations at the Exchange are confirmed.

Before starting the verification procedure it is required to go to the tab "Identification" / "Identity authentication", press one of the buttons - "Increase" (opposite the access level) or "Start verification" (Figure 6).



Identity verification

Access level — Bronze Increase

Please provide the following information to verify your identity, increase Access level and expand your limits.

Information	Status
Personal information	verified/not verified
Address	verified/not verified
Documents	verified/not verified
Photo and agreements	verified/not verified
Source of income	verified/not verified

Start verification

Figure 6 - Proceeding to the verification procedure

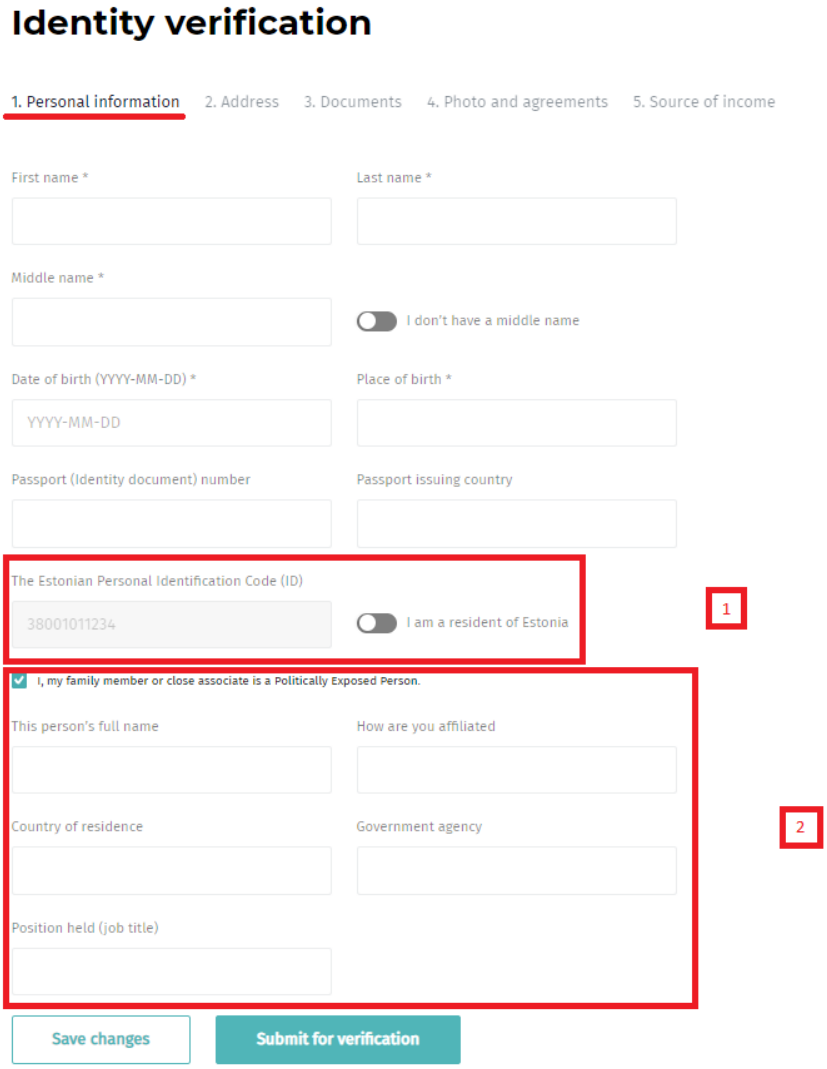
<u>The verification procedure comprises the following steps</u>:

- *Editing the personal data of the user:*

1) To input the values to the fields Name and Surname, date and place of birth in the tab "Personal information".

For the residents of Estonia it is required to indicate the personal identification code (ID).

For the purpose to observe the legislation of the EU member countries, it is required to fill in the section, which is accessible, when ticking the point of “I, my family member or relative, being the politically meaningful person”, if the said persons are employed by the government authorities or state administrative bodies. In the entry boxes of the section to input name of the contact person, name of country, state institution, title and degrees of kin (Figure 7):



Identity verification

1. Personal information 2. Address 3. Documents 4. Photo and agreements 5. Source of income

First name *

Last name *

Middle name *

I don't have a middle name

Date of birth (YYYY-MM-DD) *

YYYY-MM-DD

Place of birth *

Passport (Identity document) number

Passport issuing country

The Estonian Personal Identification Code (ID)

38001011234

I am a resident of Estonia

☑ I, my family member or close associate is a Politically Exposed Person.

This person's full name

How are you affiliated

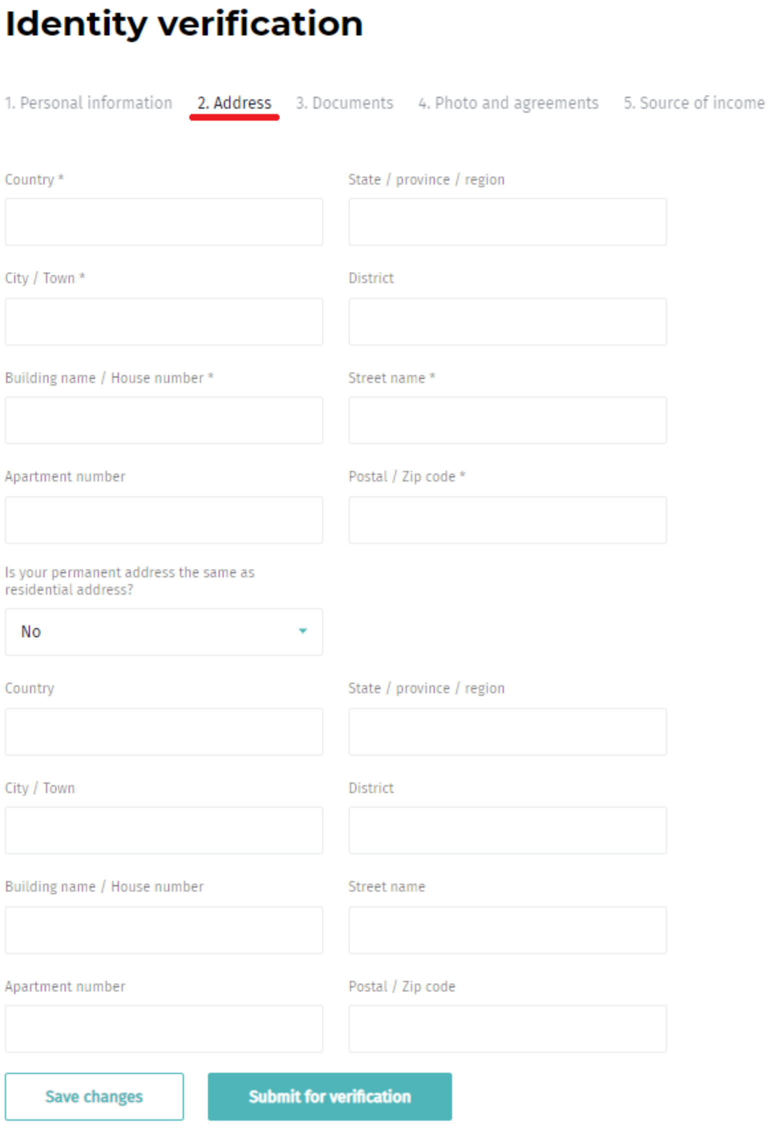
Country of residence

Government agency

Position held (job title)

Figure 7 - Tab “Personal information” in the editing format

2) To indicate the actual residence and registration addresses in the tab “Address” (Figure 8):



Is your permanent address the same as residential address?
No

Figure 8 - Tab “Address” in the editing format

- *Adding scanned copies of the documents:*

3) In the tab “Documents” to upload the scanned copies making use of button “Select”; technical characteristics of the documents are described in the section “Requirements to the documents”.

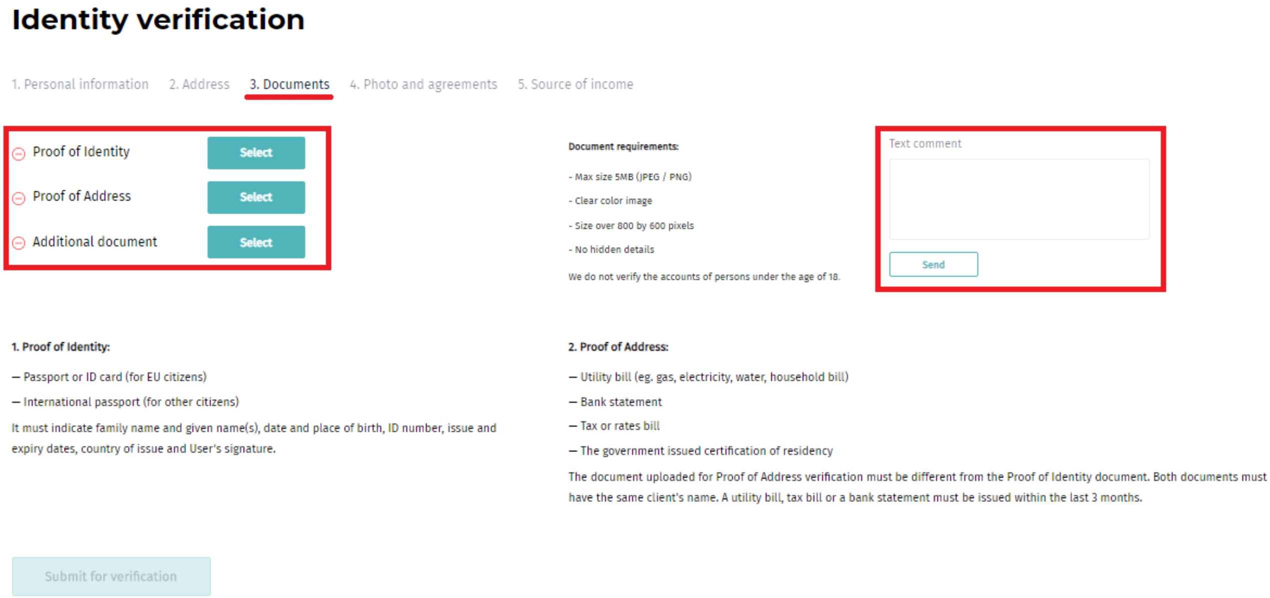
For the purpose of the client’s personal identification, it is required to provide the passport / identity card for the EU citizens or the information according to the international passport for the other countries’ citizens.

For the purpose of the user address verification one of the following documents to be selected:

- utility bill (for the last 3 months);
- bank statement (for the last 3 months);
- certificate on payment of taxes and duties (for the last 3 months);
- document certifying the residence issued by the state authority.

Maximum size of each uploaded file is 5 MB.

Additional information regarding the submitted scanned copies can be provided in the section “Comments” (Figure 9):



Document requirements:

- Max size 5MB (JPEG / PNG)

- Clear color image

- Size over 800 by 600 pixels

- No hidden details

We do not verify the accounts of persons under the age of 18.

1. Proof of Identity:

— Passport or ID card (for EU citizens)

— International passport (for other citizens)

It must indicate family name and given name(s), date and place of birth, ID number, issue and expiry dates, country of issue and User's signature.

2. Proof of Address:

— Utility bill (eg. gas, electricity, water, household bill)

— Bank statement

— Tax or rates bill

— The government issued certification of residency

The document uploaded for Proof of Address verification must be different from the Proof of Identity document. Both documents must have the same client's name. A utility bill, tax bill or a bank statement must be issued within the last 3 months.

Figure 9 - Tab “Documents” in the editing format

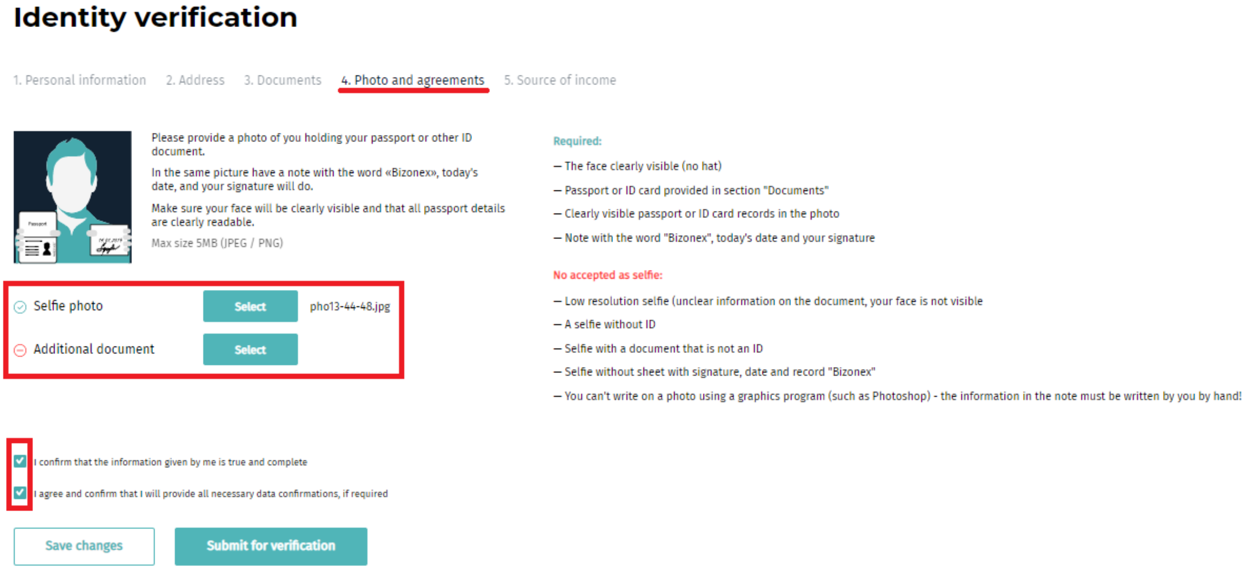
4) In the tab “Photos and consents” it is required to add the photo of the user by means of button “Select”, the list of requirements regarding its arrangement is given in the right part of the screen.

The scanned copy should contain the following elements (Figure 10):

- user photo;
- passport / travel photo;
- inscription with word “Bizonex”;
- current date;
- user’s signature made by hand.

Maximum size of each uploaded file is 5 MB.

Before saving the changes, the boxes “I confirm that the information submitted by me is authentic and complete” and “I agree and confirm that I will provide all the data proofs necessary, if required” should be ticked.



Please provide a photo of you holding your passport or other ID document.

In the same picture have a note with the word «Bizonex», today's date, and your signature will do.

Make sure your face will be clearly visible and that all passport details are clearly readable.

Max size 5MB (JPEG / PNG)

Required:

— The face clearly visible (no hat)

— Passport or ID card provided in section "Documents"

— Clearly visible passport or ID card records in the photo

— Note with the word "Bizonex", today's date and your signature

No accepted as selfie:

— Low resolution selfie (unclear information on the document, your face is not visible

— A selfie without ID

— Selfie with a document that is not an ID

— Selfie without sheet with signature, date and record "Bizonex"

— You can't write on a photo using a graphics program (such as Photoshop) - the information in the note must be written by you by hand!

☑ I confirm that the information given by me is true and complete

☑ I agree and confirm that I will provide all necessary data confirmations, if required

Figure 10 - Section “Photos and consents” in the editing format

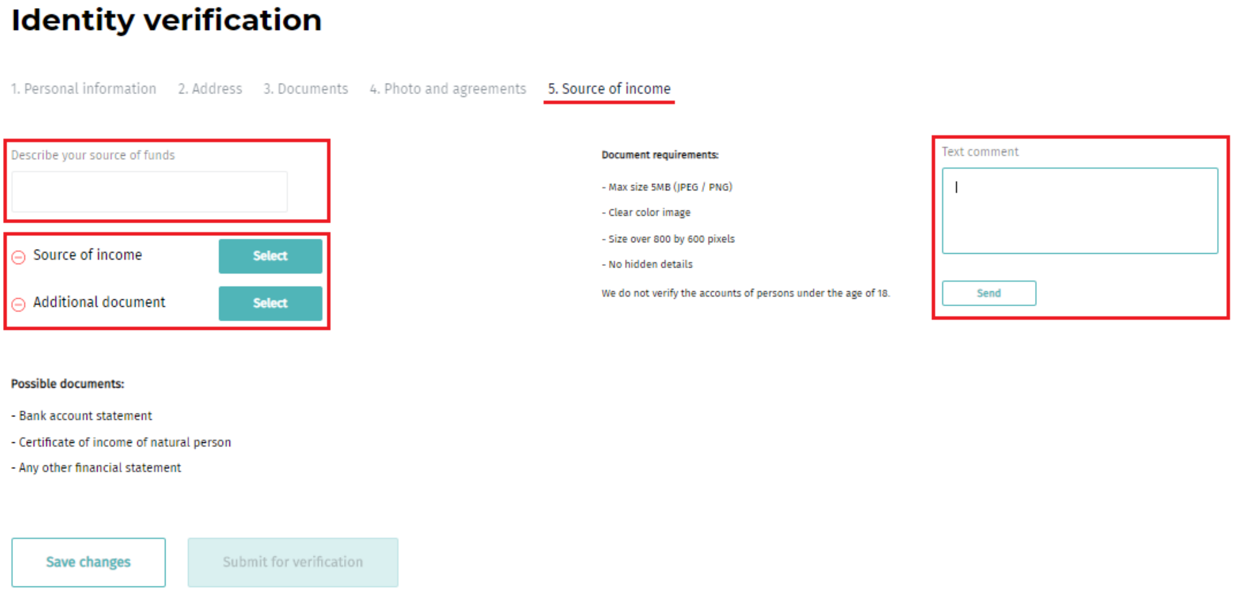
5) For making the operations at the Exchange it is required to submit the authentic data regarding the declared income source of the user having uploaded the scanned copies by means of the “Select” button.

For the purpose of the user income source verification, one of the following documents to be selected (Figure 11):

- bank account statement;
- income certificate of the individual;
- any other financial report.

Maximum size of each uploaded file is 5 MB.

Additional information regarding the submitted scanned copies can be provided in the section “Comments”.



Identity verification

Describe your source of funds

Source of income

Additional document

Document requirements:

- Max size 5MB (JPEG / PNG)
- Clear color image
- Size over 800 by 600 pixels
- No hidden details

We do not verify the accounts of persons under the age of 18.

Text comment

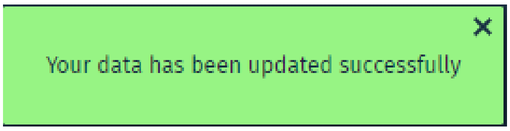
Possible documents:

- Bank account statement
- Certificate of income of natural person
- Any other financial statement

Figure 11 - Tab “Income source” in the editing format

- *Submission of information for checking:*

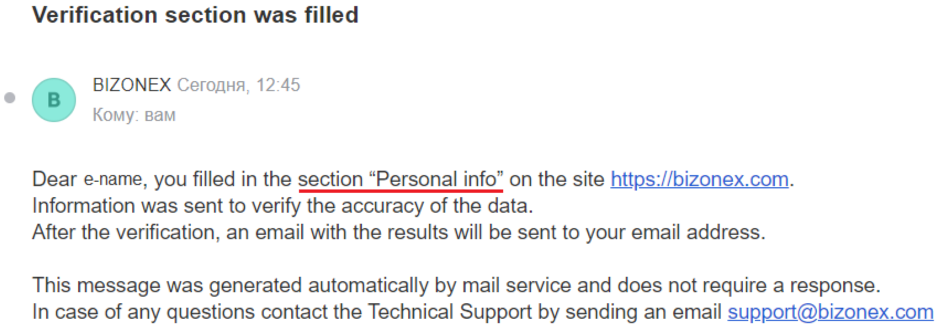
6) In order to transfer the data to server, press the button “Save changes”, in the lower part of the screen, the message will be indicated about successful uploading of information (Figure 12).



Your data has been updated successfully

Figure 12 - Information message, when saving the data

7) After sending the information to the expert for verification (button “Send for verification”) the e-mail will be sent to the user’s e-mail address about the verification of the data authenticity (Figure 13); the section will not be accessible for further editing.



Verification section was filled

B BIZONEX Сегодня, 12:45
Кому: вам

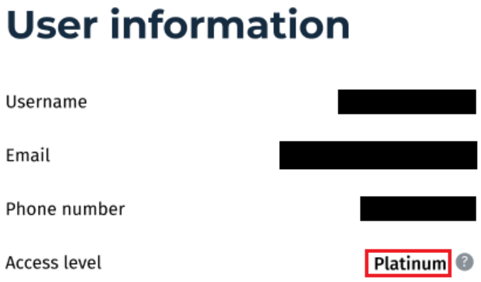
Dear e-name, you filled in the section “Personal info” on the site https://bizonex.com.
Information was sent to verify the accuracy of the data.
After the verification, an email with the results will be sent to your email address.

This message was generated automatically by mail service and does not require a response.
In case of any questions contact the Technical Support by sending an email support@bizonex.com

Figure 13 - E-mail about the sending of one of the “Verification” sections for verification of the data authenticity

- *Verification and confirmation of information that have been uploaded by the user (is performed by the experts of BIZONEX Exchange).*

8) When the data authenticity is confirmed, the user receives new access level (Figure 14); for details of the access level refer to section "AML / KYC Policy" at the page of the Exchange or KYC manual in Zendesk.



User information

Username [redacted]

Email [redacted]

Phone number [redacted]

Access level Platinum

Figure 14- Section "User profile" after completion of the verification procedure

When amending the documents or in case any questions occur, please contact the "Technical support service" at: support@bizonex.com.

The information can be additionally inquired via e-mail address of the user (account registration address) in case there are any suspicions regarding the authenticity of the documents submitted.

The terms for carrying out the verification for the first / repeated procedure are determined by the number of applications and quality of the documents submitted.

NOTE**.** Before the moment of verification procedure, the user information can be changed; after carrying out the verification, the "Verification" section is not accessible for editing.

1.3 Functional purpose of Api-keys (Token)

API-keys (Token) are the access keys to the personal information of the user, which are given to the program - bot so that to perform a number of operations as a user related to reading and writing the data at the BIZONEX Exchange.

[redacted] requests per second are performed at the Exchange, the request can include several operations, indicated in the user's.

When being used, the Token is subjected to verification of the legitimacy of application (compliance of the API-key used to the login) and authenticity of the key (matching of the pair of public and secret keys). When the usage inauthenticity is determined, the Token can be blocked. The unblocking is performed in case the user - key-owner contacts the "Technical support service".

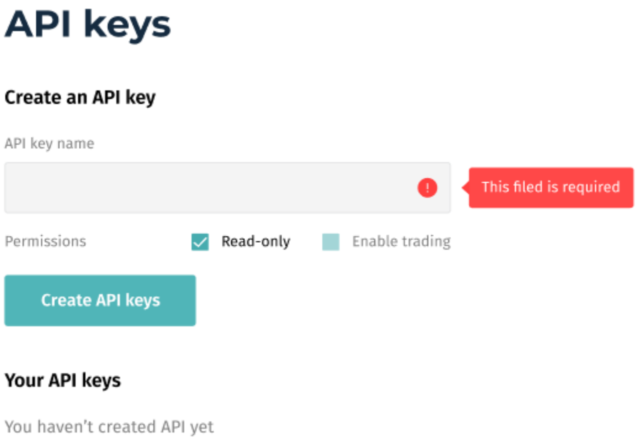
NOTE: The user is fully responsible for transferring the public / secret keys to the third persons and bears financial risks when performing any operations by the third persons making use of its API-key. Should the fact of breach the Token usage be established, the issued and pending orders can be blocked, the funds on operations can be frozen until the legitimacy of the completed transactions is established.

1.4 Creation of Token

In order to create new Api-key the user is required to perform the following actions:

- *To set the access level for bot-program by new key in the "API" section* (Figure 15)*:*

1) To input the name of the new Token (login).
2) To define the authorization level (read only, enable trade).
3) Press button "Create API".



API keys

Create an API key

API key name

This filed is required

Permissions ☑ Read-only □ Enable trading

Create API keys

Your API keys

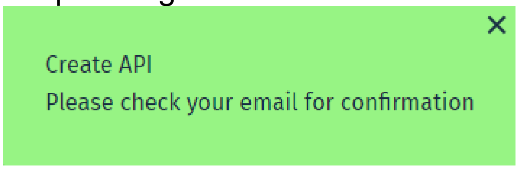
You haven't created API yet

Figure 15 - Form of creation of new Api (Token)

NOTE: Access level “bronze” enables to create the Token with authorization level “read only”, the users with higher access levels (authorization type - “enable trade”) can perform the trade operations by means of API-keys.

- *To read the information message from the letter sent for confirmation of creation of the new Token*.

Message about sending of e-mail to the user registration address will be displayed on the screen after pressing the “Create API” button (Figure 16).



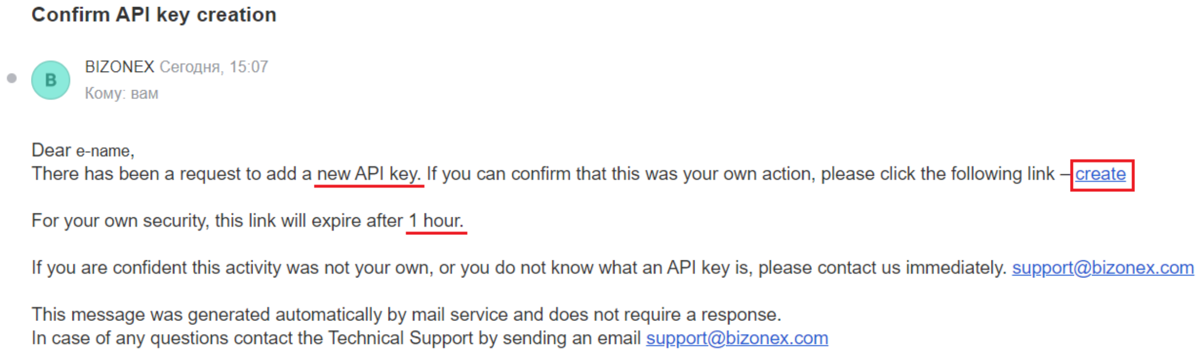
Create API

Please check your email for confirmation

Figure 16 - Information message at creation of new Api (Token)

- *To open the email, which contains the link for creation of new Api (Token) and information about the link validity period*.

To go by the link in the letter before the established time (1 hour) expires. Later on the link will not be accessible, the actions for creation of new Api-key should be repeated (Figure 17).



Confirm API key creation

BIZONEX Сегодня, 15:07

Кому: вам

Dear e-name,

There has been a request to add a new API key. If you can confirm that this was your own action, please click the following link – create

For your own security, this link will expire after 1 hour.

If you are confident this activity was not your own, or you do not know what an API key is, please contact us immediately. support@bizonex.com

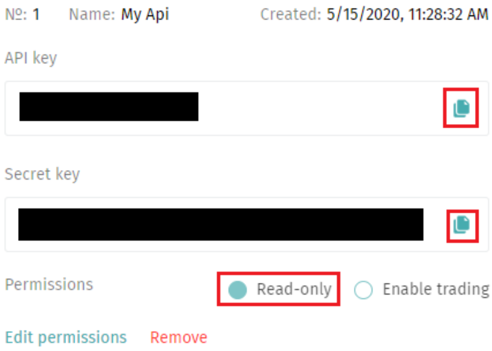
This message was generated automatically by mail service and does not require a response.

In case of any questions contact the Technical Support by sending an email support@bizonex.com

Figure 17 - Electronic message with the link for creation of Api-key

- *To set the authorizations of Api (Token).*

After going by the link in the “API” section, new Token will be created, the secret key will be shown until the page is updated (Figure 18).



№: 1 Name: My Api Created: 5/15/2020, 11:28:32 AM

API key

[redacted]

Secret key

[redacted]

Permissions ● Read-only ○ Enable trading

Figure 18 - New Token

API and its secret key are copied, when pressing the icon 📋 in the right part of the key box.

After creation, the user can change the type of authorization for each API-key (Figure 19):

1) To press button “Change authorization” under the selected API.

2) To set the type of authorization by ticking “Read only” or “Enable trade”.

3) To press button “Save” for changing the authorization or “Cancel” for keeping the previous settings.

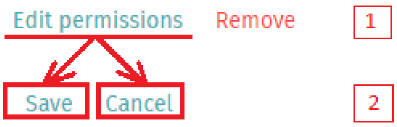
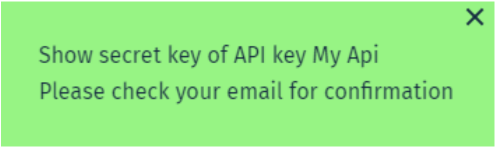


Figure 19 – Change of authorization type for Token

1.5 Opening of Token

In order to display the secret Api-key the user is required to perform the following actions:

- *To press the icon 👁 in the right part of the box “Secret key” of the selected “API”.* The information message about sending the e-mail to the user registration address to confirm the completion of the operation will be displayed on the screen (Figure 20).



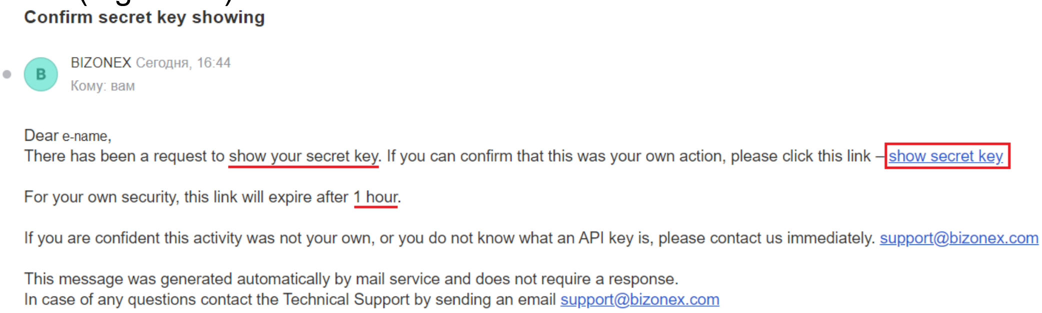
Show secret key of API key My Api

Please check your email for confirmation

Figure 20 - Information message about sending the letter

- *To open the email, which contains the link for opening of the Token and information about the link validity period*.

To go by the link in the letter before the established time (1 hour) expires. Later on the link will not be accessible, the actions for opening the secret Token key should be repeated (Figure 21).



Confirm secret key showing

B BIZONEX Сегодня, 16:44

Кому: вам

Dear e-name,

There has been a request to show your secret key. If you can confirm that this was your own action, please click this link – show secret key

For your own security, this link will expire after 1 hour.

If you are confident this activity was not your own, or you do not know what an API key is, please contact us immediately. support@bizonex.com

This message was generated automatically by mail service and does not require a response.

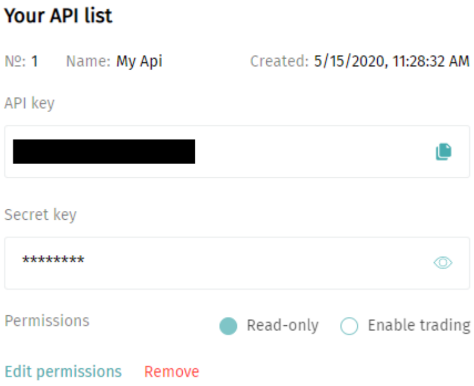
In case of any questions contact the Technical Support by sending an email support@bizonex.com

Figure 21 - Electronic message with the link for opening the secret Token key

After going by the link, the secret Token key will be displayed in the “API” section, which is available for reading till the moment of page updating.

The appearance of “API” form of the Token created contains the following elements (Figure 22):

1 Token serial number.
2 Token name – login.
3 Date and time of creation.
4 API-key – code of the public key with the possibility of its copying (📋).
3 Secret key – code of the secret key with the possibility of its copying (📋) or opening (👁).
5 Type of authorization - “Read only” or “Enable trade”.
6 Change of authorization type.
7 Deletion of token.



Your API list

№: 1 Name: My Api Created: 5/15/2020, 11:28:32 AM

API key

[redacted]

Secret key

****

Permissions ● Read-only ○ Enable trading

Edit permissions Remove

Figure 22 - Appearance of the “API” form of the Token created

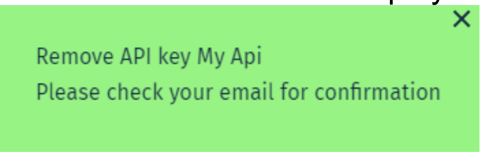
NOTE: Value of boxes “Public key” and “Secret key” shall not be provided to the third persons.

1.1 Deletion of Token

In order to delete the Api-key the user is required to perform the following actions:

- To press the “Delete” button under the selected Token.

The information message about sending the e-mail to the user registration address to confirm the deletion of the Token will be displayed on the screen (Figure 23).

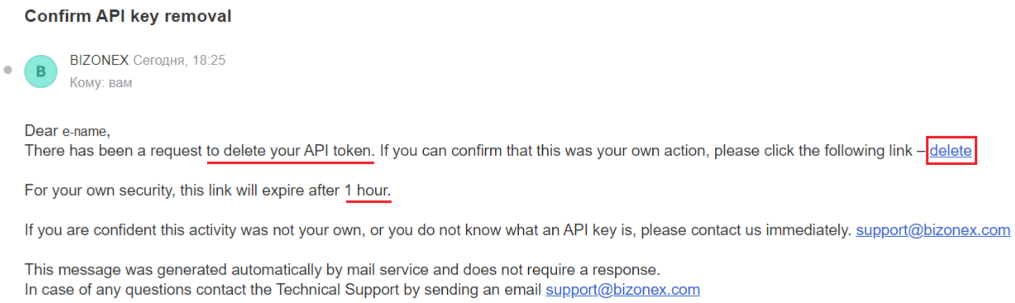


Remove API key My Api
Please check your email for confirmation

Figure 23 - Information message about sending the letter

- To open the email, which contains the link for confirmation of deletion of the Token and information about the link validity period.

To go by the link in the letter before the established time (1 hour) expires. Later on the link will not be accessible, the actions for deletion of Token should be repeated (Figure 24).



Confirm API key removal

BIZONEX Сегодня, 18:25
Кому: вам

Dear e-name,
There has been a request to delete your API token. If you can confirm that this was your own action, please click the following link – delete

For your own security, this link will expire after 1 hour.

If you are confident this activity was not your own, or you do not know what an API key is, please contact us immediately. support@bizonex.com

This message was generated automatically by mail service and does not require a response.
In case of any questions contact the Technical Support by sending an email support@bizonex.com

Figure 24 - Electronic message with the link for deletion of Api-key

2 Description of queries

Queries are written in JavaScript using GraphQL and JSON.

GraphQL is a query language for your API, and a server-side runtime (program execution period) for executing queries. The query syntax is passed to the server in Graphql, and if data cannot be output in requested format, an error message will be displayed.

JSON is a format for query data transmission. API is based on JSON RPC of Websocket Protocol. Websocket is an HTTP extension allowing the applications to maintain multi-user interaction in real time, and using this approach asynchronous queries are sent to the server, and responses are processed using callback functions.

Before executing queries, do the following:

1. Check and, if necessary, change the format of "Date and Time" parameter on the user's PC. Settings should be synchronized with the server time (time synchronization type by Internet).
2. Enter public and private Token keys in "apiKey" and "apiSecret" fields. Keys receipt is described in section 1 of the current documentation (for more details, see sections 1.4, 1.5, and 1.6).

To complete query, do the following:

1. For HTTP Protocol, comment on queries that will not be executed in the current session (script contains 10 queries in section about query).

For Websocket Protocol specify query to be executed in string:

```
const request = requestTable.<Query name>
```

Until connection closure, subscription remains active and is waiting for server responses. To close connection, click "Ctrl" + "C".

2. Select data output type (for more details, see section 2.3).
3. After making any changes, click *CTRL* + *"S"* to save the latest changes to the query.
4. Write query text; query format depends on the Protocol type.

Query structure by the HTTP Protocol:
{query type: `{method name {list of keys {list of parameters}}}`},
where "list of parameters" is an optional field.
Query example under the HTTP Protocol:

```
query: `{marketList{name, stock, money, fee_prec, stock_prec, money_prec,
min_amount}}`
```

Query structure by the Websocket Protocol:
{query type: `subscription + method name (variable type declaration: string) {list of keys (reference to the variable type) {list of parameters
}}`, variable type: {name of currency or currency pair}}. The "list of parameters" is an optional field.
Query example under the Websocket Protocol:

```
query: `subscription marketUpdated($currency: String!) {
        marketUpdated(currency: $currency) {
            market
            stock
            money
            open
            high
            low
            lastPrice
            percentChange
            volume  }  }`,
```

```
            variables: { currency: 'BTC' },
```

5. The *node + "file name without extension" command* (for example, *node reqHttp*) allows performing query.

NOTE: In queries, a set of "key-value" pairs can be shown in single quotes of two types – "` `" or " ' ' ". "Text" type values are passed to the scripts in double quotes–" ", and values of the "number" type or "boolean" type are passed without quotes (for example, "text" type - "BTCUSDT" and "number" type - 1).

2.1 Queries structure

Query specification includes several sections, each of which has its own functional purpose:

1) Connection settings for the HTTP and Websocket Protocols. This section contains data about connected libraries/modules and defines the format for the information transmission between the client and the server when executing queries.

2) Query transmission. This section contains data transmission settings, a set of parameters and their values for the information display in the specified format.

3) List of queries. This section includes a full list of methods with input and output parameters listed (for more details, see section 2.2).

4) Information display in web-console. This section describes the format for the information display in response to the query made by the server (for more details, see section 2.3).

5) Description of errors. This section contains a list of possible business-logic errors or system errors when displaying information in the web-console (for more details, see section 2.4).

A detailed description of each section is provided in the text of the document below.

1) Connection settings for the HTTP and Websocket Protocols.

For more information about connection characteristics and information about files where they are used, see in Table 2.

Table 2 - Connection settings for the HTTP and Websocket Protocols

No.	Constants of the HTTP variable	Constants of the Websocket variable
1	**"crypto"** The library provides for the encryption functionality (includes a set of hashing, encryptors, decryptors, and verification functions). `const crypto = require('crypto')`	

2	**"apiKey" and " apiSecret»**
	Setting up a public-private key pair of the user's Token ("User information" tab on page http://bizonex.com/user/api, for more details, see section 1).
	To transmit query data, enter key value in single quotes (" ' ' ") in "Enter public key Token" and "Enter private key Token" fields. If Token keys (public and private) are not specified, it is impossible to execute queries of the "private method" type, but the keys entry does not affect "public method" type query.
	`const apiKey = 'Enter public key Token'`
	`const apiSecret = 'Enter private key Token'`	

3	**"nonce"** The format for time transmission in Unix, and the value is calculated in seconds. Current time on the user's PC is calculated in seconds, and when sending query the time on user's PC and on server is checked for consistency The query is executed in case of consistency, but in case of discrepancy, an error will be returned stating that the query cannot be executed. `const nonce = (Date.now() / 1000) | 0`	
4	**"request"** The library passes requests via JavaScript. `const request = require('request')`	**"request"** The library passes the name of the method executed. The list of the methods is given in section 2.2. `const request = requestTable.<Method name>`

5	**"gurl"**
	Format for the global url address transmission.
	The link text contains section - "api" and program version – "v1".
	``const gurl = `https://bizonex.com/api/v1/` ``
		«wsUrl» and «httpUrl»
		Format for the global url address transmission:
		- **wsUrl**- Websocket Protocol connection;
		- **httpUrl** - signature addition.
		The link text contains section - "api" and program version – "v1".
		`const wsUrl = 'wss://bizonex.com/api/v1'`
		`const httpUrl = 'https://bizonex.com/api/v1/'`

6	**"signature"**
	Signature format, which includes:
	- url verification;
	- time consistency set on the user's PC with the time on server;
	- JSON.stringify() method, which converts JavaScript value into a JSON string. In this case, values are substituted in the query "body".
	The scope of variable visibility – blocks of values passed in curly brackets – "{ }", and combined with "$" symbol. This type of variable becomes visible only upon declaration, and when used in a cycle, a different variable is created for each iteration.
	``const signature = `${gurl}${nonce}${JSON.stringify(body)}` ``
		"signature"
		The process of the signature format creation and declaration, which includes:
		- url verification;
		- time consistency set on the user's PC with the time on server;
		- JSON.stringify() method, which converts JavaScript value into a JSON string. In this case, values are substituted in the query "body".
		- private key Token;
		- "crypto" encryption library uses createHmac package, encrypted with "sha512" keys and secret Token key, and the signature is updated and information is processed using hex method.
		- HMAC is one of the mechanisms to verify information integrity, allowing you to ensure data transmitted or stored in unreliable environment not to be changed by the third parties.
		- key "sha512" is a hashing algorithm that is a function of the SHA-2 cryptographic algorithm. It is used in various applications related to the information protection.
		- Hex - is a 16-ary calculation system (16-bit addressing).
		Data is placed in Header, body, and query is sent.
		`const getServerSignature = ({ nonce, body, apiSecret }) => {`
		` const data = `` `${httpUrl}${nonce}${JSON.stringify(body)}` ``

7	**"shex"** Data conversion process includes the following elements: - "crypto" encryption library uses createHmac package, encrypted with "sha512" keys and secret Token key, and the signature is updated and information is processed using hex method. - HMAC is one of the mechanisms to verify information integrity, allowing you to ensure data transmitted or stored in unreliable environment not to be changed by the third parties.	

	- key "sha512" is a hashing algorithm that is a function of the SHA-2 cryptographic algorithm. It is used in various applications related to the information protection.
	- Hex - is a 16-ary calculation system (16-bit addressing).
	Data is placed in Header, body, and query is sent.
	`const shex = crypto`
	`.createHmac('sha512', Buffer.from(apiSecret, 'hex'))`
	`.update(signature)`
	`.digest('hex')`
		`const serverSignature = crypto`
		`.createHmac('sha512', Buffer.from(apiSecret, 'hex'))`
		`.update(data)`
		`.digest('hex')`
		
		`return serverSignature`
		`}`
		`const signature = getServerSignature({`
		`nonce,`
		`body: request.query,`
		`apiSecret,`
		`})`

8	**"options"**
	A set of parameters added to the query:
	- content of HTTP query is encrypted as a query string;
	- link to url address;
	- query content is transmitted to the query "body". The header contains the following indicators: current date and time, public and private keys from Token, signature in the format specified when defining shex constant.
	- information about data transmission type (in JSON format).
	`const options = {`
	`contentType: 'application/ json',`
	``url: `https://bizonex.com/api/v1/`,``
	`headers: {`
	`nonce,`
	`apiKey,`
	`signature: shex,`
	`},`
	`body,`
	`json: true,`
	`}`
		"options"
		Format for a set of parameters output added to the query:
		- query header (time transfer format in Unix, public key Token, signature);
		- results of query execution.
		`const options = {`
		`headers: { nonce, apikey, signature },`
		`body: request.query,`
		`}`

9	-	**"WebSocketLink"** (function from "apollo-link-ws" package) Library for Graphql-client creation that uses Websocket Protocol. `const { WebSocketLink } = require('apollo-link-ws')`
10	-	**"gql"** The library is used for a Graphql-query analysis (parsing) when determining configuration file settings. Additionally, check library connection containing software for graphql-tag – Graphql. `const gql = require('graphql-tag').default`
11	-	**"SubscriptionClient"** (function from "subscriptions-transport-ws" package) Library for the client creation preparing and transmitting information received from server using Websocket Protocol subscriptions. `const { SubscriptionClient } = require('subscriptions-transport-ws')`
12	-	**"ws"** Library for Websocket-connection package. `const ws = require('ws')`
13	-	**"execute"** (function from "apollo-link" package)

		Library for the query functions. const { execute } = require('apollo-link')
14	-	**"requestTable"** The structure contains a list of methods described in details in section 2.2.

15	-	**"createSubscriptionObservable"** and **"subscribe"**
		The process of format creation and declaration of the Protocol launch monitoring events (subscriptions) includes:
		 - appeal to the library WebSocketLink**;**
		- appeal to the library SubscriptionClient;
		- Websocket wsUrl Protocol connection;
		- parameters transmission.
		The client is prepared to send a Graphql-query to the server and query execution.
		Unlike the HTTP Protocol, data is received via the Websocket Protocol when an event the subscription to which was made occurs. Until connection closure, subscription remains active and is waiting for server responses.
		const createSubscriptionObservable = (wsUrl, query, variables) => {
		    const link = new WebSocketLink(
		        new SubscriptionClient(
		            wsUrl,
		            {
		                connectionParams: options,
		            },
		            ws
		        )
		    )
		
		    return execute(link, {
		        query,
		        variables,
		    })
		}
		
		const subscribe = () => {
		    const subscriptionObservable = createSubscriptionObservable(
		        wsUrl,
		        gql`
		            ${request.query}
		        `,
		        request.variables
		    )
		subscribe()

2) Query transfer

A list of methods is transferred in query allowing to perform API transactions on the Stock Exchange. This document describes 19 HTTP methods and 7 Websocket methods (for more details, see section 2.2).

Only the running query should be active, and all other queries should be commented. To execute multiple methods simultaneously, queries should be transmitted to a single query.

Example of two queries combination into one:

- query 1;

```
query: '{marketList{name, stock, money, fee_prec, stock_prec, money_prec, min_amount}}',
```

- query 2;

```
query: '{orderBook(market: "BTCUSDT", limit: 50, precision: "10"){asks{price, amount}, bids{price, amount}}}',
```

- query 3 combining queries 1 and 2 (the following characters were removed between them to combine queries: “}`,    query: `{").

```
query: `{marketList{name, stock, money, fee_prec, stock_prec, money_prec, min_amount}

orderBook(market: "BTCUSDT", limit: 50, precision: "10"){asks{price, amount}, bids{price, amount}}}`,
```

For Websocket-queries, a requestTable structure has been created that contains a full list of queries, and the query name being executed shall be transmitted in request variable.

```
const request = requestTable.<Method name>
```

Parameters for getting values can be transmitted in any order, the number of parameters is unlimited, and examples of entries are as follows:

```
tradeHistory(market: "BTCUSDT", limit: 10, offset: 100){price, time}
tradeHistory(market: "BTCUSDT", limit: 10, offset: 100){id, time, price, amount, type}
```

3) List of queries

Full list of methods with input and output parameters. Detailed information is provided in section 2.2, and queries texts are provided in Annexes A and B.

4) Information display in web-console.

When transmitting query data in POST format, Node.js is addressed to a third-party API and gets a response. The response format is an error message or response text.

Query results are displayed as messages to the web-console with parameters according to the format specified, as illustrated in "Output data" section (for more details, see section 2.3).

An example shows data transmission format for the HTTP queries:

```
request.post(options, (error, response, body) => {"Specify data output form"})
```

NOTE. Data output form is enclosed in curly brackets ("{}").

5) Errors description

When executing queries, you may get two types of errors when displaying information in the web-console:

- business-logic errors;
- system errors.

For more details, see section 2.4.

2.2 Methods description

To obtain values by user requests, 26 methods are used for API, for more details, see Table 3:

- 19 HTTP methods (public – 7, private – 12);
- 7 Websocket methods (public – 5, private – 2).

Table 3 - List of HTTP and Websocket methods:

HTTP		Websocket	
public methods	**private methods**	**public methods**	**private methods**
chart (Table 8)	**balance** (Table 4)	**chartUpdated** (Table 9)	**balanceUpdated** (Table 6)
currencyList (Table 10)	**balanceHistory** (Table 5)	**dealsUpdated** (Table 11)	**orderUpdated** (Table 18)
marketList (Table 12)	**cancelOrder** (Table 7)	**marketUpdated** (Table 14)	
marketToday (Table 13)	**orderHistory** (Table 17)	**orderBookUpdated** (Table 16)	
orderBook (Table 15)	**pendingOrders** (Table 19)	**stateUpdated** (Table 26)	
tradeHistory (Table 27)	**pendingOrderDetail** (Table 20)		
walletCommissions (Table 28)	**pendingSummary** (Table 21)		
	putLimitOrder (Table 22)		
	putMarketOrder (Table 23)		
	putStopLimitOrder (Table 24)		
	putStopOrder (Table 25)		
	walletQuery (Table 29)		

Methods include the query text and its parameters and are transmitted in "key – value" format. Query parameters are an optional attribute.

Only one request can be made for HTTP, and all other requests should be commented. Example for multiple queries combination into one is given in paragraph "Query transmission as a "key-value" pair" (for more details, see section 2.1).

Structure for transmitting HTTP methods:

```
: `{Transmit query and its parameters}`, }
```

For Websocket, "requestTable" constant contains a full list of methods, and the "request" constant contains the name of the method being called.

Until connection closure, subscription remains active and is waiting for server responses. To close connection, click *"Ctrl"* + *"C"*.

Structure for transmitting Websocket methods:

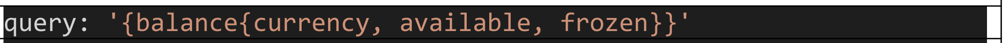
```

        query: `{Send query}`,
        variables: {Transmit query parameters} <Method Name>
```

NOTE. Query text and its parameters are enclosed in curly brackets ("{}").

Table 4 – Method balance (User's balance)

Method name	**balance**
Method description	**User balance**
Protocol type	HTTP

Method type	Private method
Incoming parameters	—
Outgoing parameters	**currency** - currency name entering the market; **available** - available user's balance; **frozen** - reserved balance for orders placed.
Example of use	 query: '{balance{currency, available, frozen}}'

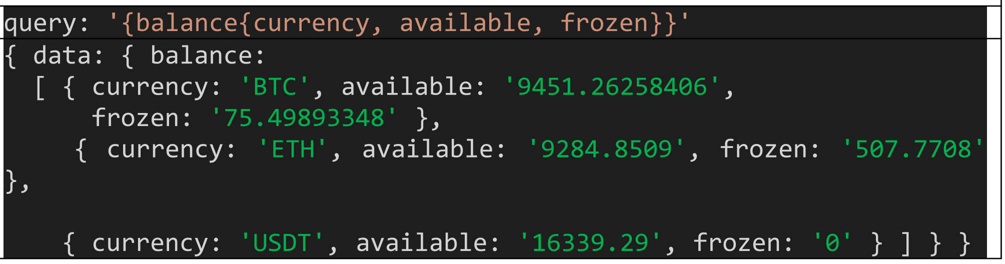
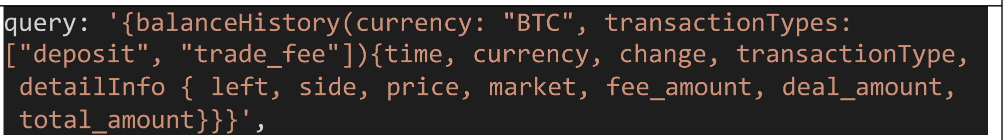
Example of response	 { data: { balance:
	  [{ currency: 'BTC', available: '9451.26258406',
	      frozen: '75.49893348' },
	     { currency: 'ETH', available: '9284.8509', frozen: '507.7708'
	},
	
	    { currency: 'USDT', available: '16339.29', frozen: '0' }] } }

Table 5 – BalanceHistory Method (History by user's balance)

Method name	**balanceHistory**
Method description	**History by user`s balance**
Protocol type	HTTP
Method type	Private method
Incoming parameters	**currency** is a currency name entering the market; **transactionTypes** is a type of currency transactions (trade_fee – "trade fee", trade – "trade", deposit – "deposit", discount – "discount", withdrawal - "funds withdrawal", reward - "reward"). Indicator can contain either one or several parameters.
Outgoing parameters	**time** is a query completion time (in Unix Timestamp format); **currency** is a currency name entering the market; **change** is a change in price for the currency pair selected at the moment to the price for the currency pair selected at the time of Stock Exchange opening for the last 24 hours in percentage; **transactionType** is a type of currency transaction; **left** is a not redeemed balance of order relative to the total amount; **side** is an order status (limit order - 0, purchase order - 1, sell order - 2); **price** is a price per unit of currency when buying / selling order; **market** - **fee_amount** is a fee charged for the number of currency units displayed; **deal_amount** is a number of currency units displayed for transaction completed; **total_amount** is a total amount for transaction completed.
Example of use	 query: '{balanceHistory(currency: "BTC", transactionTypes: ["deposit", "trade_fee"]){time, currency, change, transactionType, detailInfo { left, side, price, market, fee_amount, deal_amount, total_amount}}}',

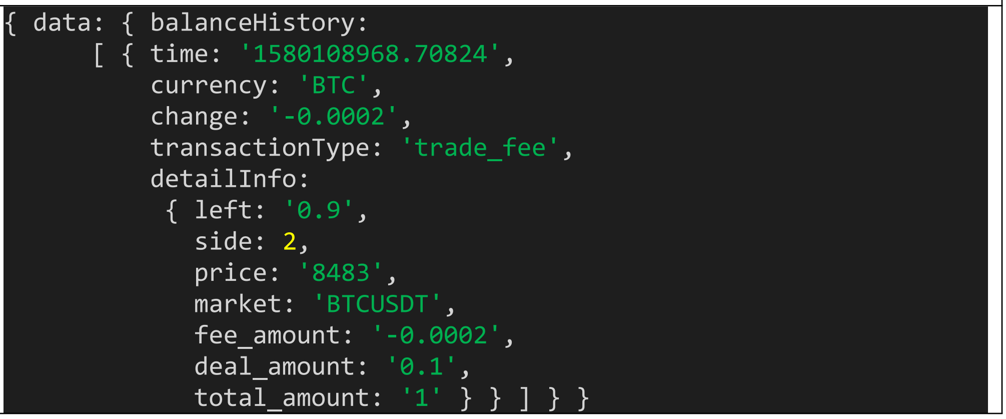
Example of response	 { data: { balanceHistory:
	      [{ time: '1580108968.70824',
	          currency: 'BTC',
	          change: '-0.0002',
	          transactionType: 'trade_fee',
	          detailInfo:
	           { left: '0.9',
	             side: 2,
	             price: '8483',
	             market: 'BTCUSDT',
	             fee_amount: '-0.0002',
	             deal_amount: '0.1',
	             total_amount: '1' } }] } }

Table 6 – BalanceUpdated Method (Subscription to the user's balance update)

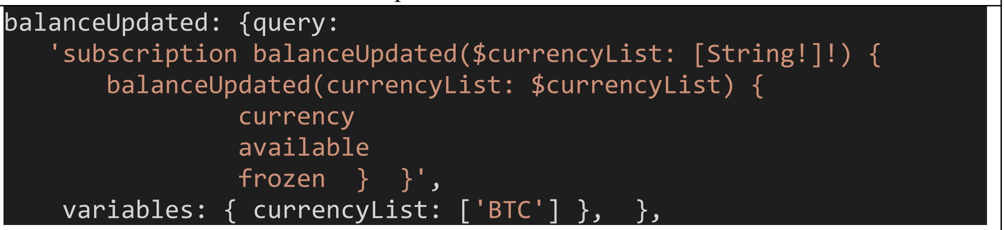
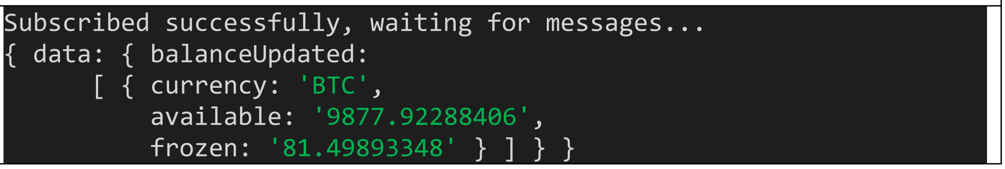
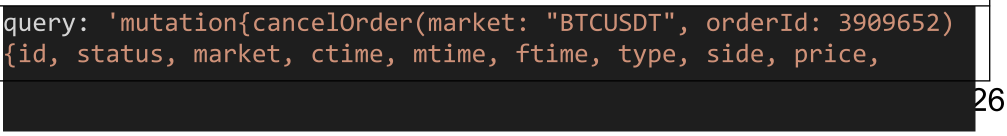
<table>
<tr><td><b>Method name</b></td><td><b>balanceUpdated</b></td></tr>
<tr><td><b>Method description</b></td><td><b>Subscription to the user's balance update</b></td></tr>
<tr><td><b>Protocol type</b></td><td>Websocket</td></tr>
<tr><td><b>Method type</b></td><td>Private method</td></tr>
<tr><td><b>Incoming parameters</b></td><td><b>currencyList</b> is a List of currencies by which changes should be tracked</td></tr>
<tr><td><b>Outgoing parameters</b></td><td><b>currency</b> - currency name entering the market;
<b>available</b> - available user's balance;
<b>frozen</b> - reserved balance for orders placed.</td></tr>
<tr><td><b>Example of use</b></td><td>
<pre>
balanceUpdated: {query:
   'subscription balanceUpdated($currencyList: [String!]!) {
       balanceUpdated(currencyList: $currencyList) {
                currency
                available
                frozen   } }',
    variables: { currencyList: ['BTC'] }, },
</pre></td></tr>
<tr><td><b>Example of response</b></td><td>
<pre>
Subscribed successfully, waiting for messages...
{ data: { balanceUpdated:
      [{ currency: 'BTC',
          available: '9877.92288406',
          frozen: '81.49893348' }] } }
</pre></td></tr>
</table>

Table 7 – CancelOrder Method (Cancellation of the order placed)

<table>
<tr><td><b>Method name</b></td><td><b>cancelOrder</b></td></tr>
<tr><td><b>Method description</b></td><td><b>Cancellation of the order placed</b></td></tr>
<tr><td><b>Protocol type</b></td><td>HTTP</td></tr>
<tr><td><b>Method type</b></td><td>Private method</td></tr>
<tr><td><b>Incoming parameters</b></td><td><b>market</b> is a market name;
<b>orderId</b> is an ordinal number of order.</td></tr>
<tr><td><b>Outgoing parameters</b></td><td><b>id</b> is an order ID;
<b>status</b> is an order execution status ("pending" - pending execution, "updated" - partially executed, "finished" - fully executed, "canceled" - cancelled);
<b>market</b> is a market name;
<b>ctime</b> is a time when orders were created (in Unix Timestamp format);
<b>mtime</b> is a time of the last order change (in Unix Timestamp format);
<b>ftime</b> is an order closing time (in Unix Timestamp format);
<b>type</b> is an order type:
- "Market" - market type;
- "Good-Till-Cancel" - limited, active until manually cancelled;
- "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, and executed order part is cancelled;
- "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled.
<b>side</b> is a type of orders transactions ("asks" - sell, "birds" - buy);
<b>price</b> is a price per unit of currency when buying / selling order;
<b>amount</b> is a number of currency units displayed;
<b>taker_fee</b> is a fee charged at the moment when the user purchases another user's offer;
<b>maker_fee</b> is a fee charged at the moment when the user makes a new offer to buy / sell order;
<b>deal_fee</b> is a fee on transaction;
<b>deal_stock</b> is a transaction volume in the orders purchase;
<b>deal_money</b> is a volume of transaction in the orders selling;
<b>left</b> is a not redeemed balance of order relative to the total amount.</td></tr>
<tr><td><b>Example of use</b></td><td>
<pre>
query: 'mutation{cancelOrder(market: "BTCUSDT", orderId: 3909652)
{id, status, market, ctime, mtime, ftime, type, side, price,
</pre></td></tr>
</table>

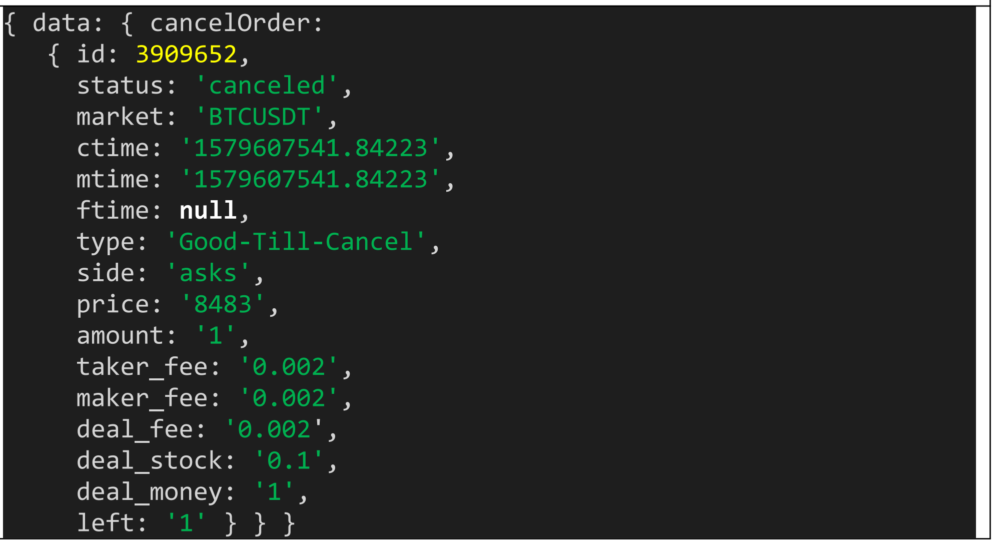
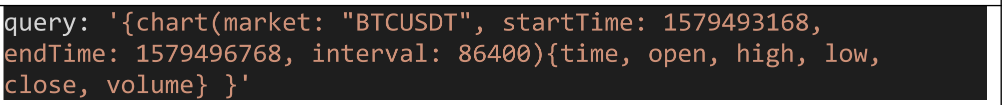
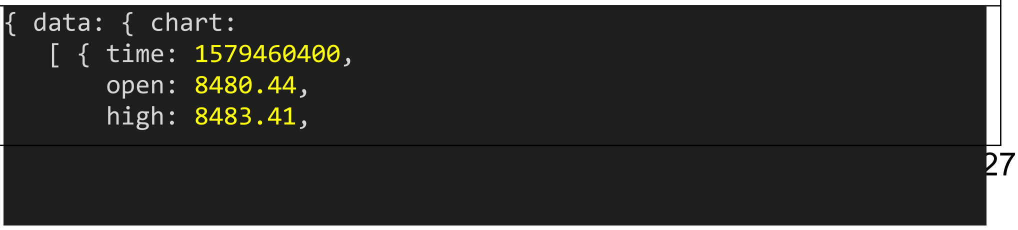
	amount,taker_fee, maker_fee, deal_fee, deal_stock, deal_money, left} }',
Example of response	 { data: { cancelOrder: { id: 3909652, status: 'canceled', market: 'BTCUSDT', ctime: '1579607541.84223', mtime: '1579607541.84223', ftime: null, type: 'Good-Till-Cancel', side: 'asks', price: '8483', amount: '1', taker_fee: '0.002', maker_fee: '0.002', deal_fee: '0.002', deal_stock: '0.1', deal_money: '1', left: '1' } } }

Table 8 - Chart Method (History of changes in the chart for the currency pair comparison by the amount of funds sold/purchased to the transaction price for the specified time period)

Method name	**chart**
Method description	**History of changes in the chart for the currency pair comparison by the amount of funds sold/purchased to the transaction price for the specified time period**
Protocol type	HTTP
Method type	Public method
Incoming parameters	**market** is a market name; **startTime** is a start time of the day (in Unix Timestamp format); **endTime** is an end time of the day (in Unix Timestamp format); **interval** is a time interval for quotations in seconds displayed on the price chart: - 30 — 30 seconds; - 60 — 1 minute; - 120 — 2 minutes; - 300 — 5 minutes; - 600 — 10 minutes; - 900 — 15 minutes; - 1800 — 30 minutes; - 3600 — 1 hour; - 14400 — 4 hours; - 86400 — 1 day; - 172800 — 2 days.
Outgoing parameters	**time** is a time when transaction was completed during the specified time period (in Unix Timestamp format); **open** is a price for the currency pair selected at the time of Stock Exchange opening; **high** is a maximum price on the date of transaction for the currency pair selected; **low** is a minimum price on the date of transaction for the currency pair selected; **close** is a price for the currency pair selected at the time of Stock Exchange closure; **volume** is a volume of trades performed by currency types.
Example of use	 query: '{chart(market: "BTCUSDT", startTime: 1579493168, endTime: 1579496768, interval: 86400){time, open, high, low, close, volume} }'
Example of response	 { data: { chart: [{ time: 1579460400, open: 8480.44, high: 8483.41,

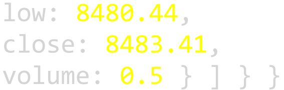
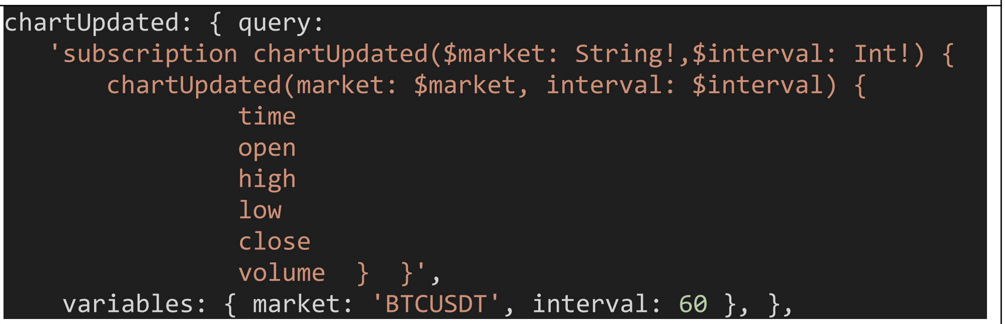
	 low: 8480.44, close: 8483.41, volume: 0.5 }] } }

Table 9 - ChartUpdated Method (Subscription to changes in the chart for the currency pair comparison by the amount of funds sold/purchased to the transaction price for the specified time period)

Method name	**chartUpdated**
Method description	**Subscription to changes in the chart for the currency pair comparison by the amount of funds sold/purchased to the transaction price for the specified time period**
Protocol type	Websocket
Method type	Public method
Incoming parameters	**market** is a market name; **interval** is a time interval for quotes in seconds displayed on the price chart: - 30 — 30 seconds; - 60 — 1 minute; - 120 — 2 minutes; - 300 — 5 minutes; - 600 — 10 minutes; - 900 — 15 minutes; - 1800 — 30 minutes; - 3600 — 1 hour; - 14400 — 4 hours; - 86400 — 1 day; - 172800 — 2 days.
Outgoing parameters	**time** is a time when transaction was completed during the specified time period (in Unix Timestamp format); **open** is a price for the currency pair selected at the time of Stock Exchange opening; **high** is a maximum price on the date of transaction for the currency pair selected; **low** is a minimum price on the date of transaction for the currency pair selected; **close** is a price for the currency pair selected at the time of Stock Exchange closure; **volume** is a volume of trades performed by currency types.

Example of use	 chartUpdated: { query:
	   'subscription chartUpdated($market: String!,$interval: Int!) {
	       chartUpdated(market: $market, interval: $interval) {
	                time
	                open
	                high
	                low
	                close
	                volume } }',
	    variables: { market: 'BTCUSDT', interval: 60 }, },

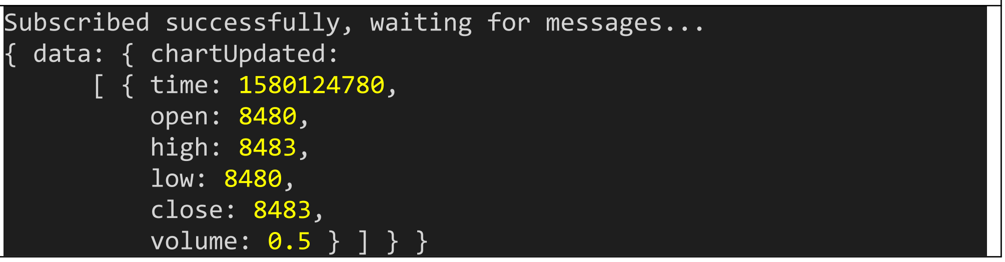
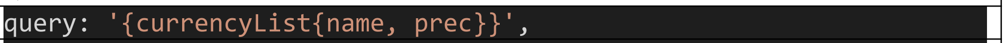
Example of response	 Subscribed successfully, waiting for messages...
	{ data: { chartUpdated:
	      [{ time: 1580124780,
	          open: 8480,
	          high: 8483,
	          low: 8480,
	          close: 8483,
	          volume: 0.5 }] } }

Table 10 - CurrencyList Method (List of currencies receipt)

Method name	**currencyList**
Method description	**List of currencies receipt**
Protocol type	HTTP

Method type	Public method
Incoming parameters	—
Outgoing parameters	**name** is a name of currency used on the Stock Exchange (cryptocurrencies and fiat money); **prec** is a number of decimal places for the currency types (cryptocurrency - 8, fiat money - 2).
Example of use	 `query: '{currencyList{name, prec}}',`

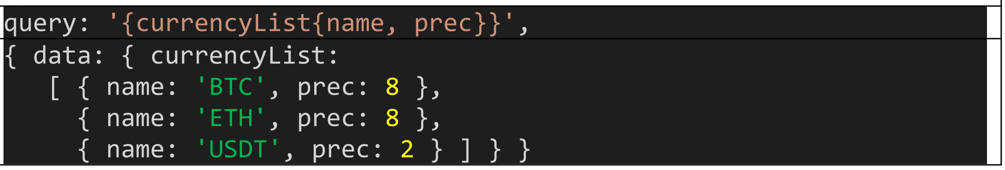
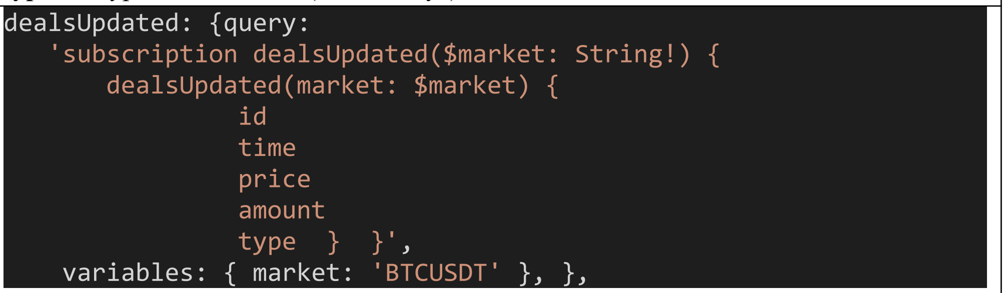
Example of response	 `{ data: { currencyList:`
	`[ { name: 'BTC', prec: 8 },`
	`{ name: 'ETH', prec: 8 },`
	`{ name: 'USDT', prec: 2 } ] } }`

Table 11 - DealsUpdated Method (Subscription to the list update for the sale/purchase transactions completed)

Method name	**dealsUpdated**
Method description	**Subscription to the list update for the sale/purchase transactions completed**
Protocol type	Websocket
Method type	Public method
Incoming parameters	**market** is a market name;
Outgoing parameters	**id** is a transaction ID; **time** is a transaction completion time (in Unix Timestamp format); **price** is a price per unit of currency when buying / selling order; **amount** is a number of currency units displayed; **type** is a type of transaction ("sell", "buy").

Example of use	 `dealsUpdated: {query:`
	`'subscription dealsUpdated($market: String!) {`
	`dealsUpdated(market: $market) {`
	`id`
	`time`
	`price`
	`amount`
	`type } }',`
	`variables: { market: 'BTCUSDT' }, },`

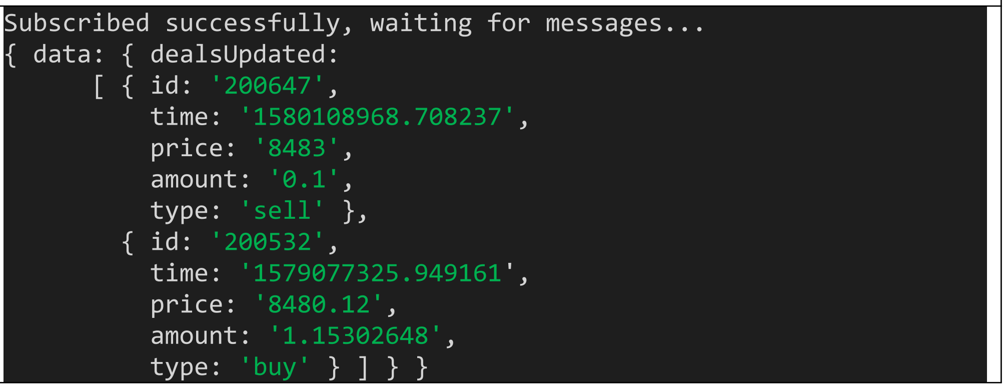
Example of response	 `Subscribed successfully, waiting for messages...`
	`{ data: { dealsUpdated:`
	`[ { id: '200647',`
	`time: '1580108968.708237',`
	`price: '8483',`
	`amount: '0.1',`
	`type: 'sell' },`
	`{ id: '200532',`
	`time: '1579077325.949161',`
	`price: '8480.12',`
	`amount: '1.15302648',`
	`type: 'buy' } ] } }`

Table 12 - MarketList Method (List of markets and their parameters receipt)

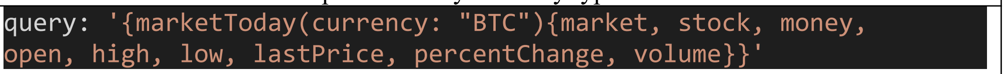
Method name	**marketList**
Method description	**List of markets and their parameters receipt**
Protocol type	HTTP
Method type	Public method
Incoming parameters	—
Outgoing parameters	**name** is a name of currency used on the Stock Exchange (cryptocurrencies and fiat money); **stock** is a type of currency purchased;

	money is a monetary equivalent; **fee_prec*** is a number of digits after decimal point for the type of "fee"; **stock_prec*** is a number of decimal places for the currency of "stock" type. When placing limit or market order, the amount value shall be rounded, and when receiving a list of orders from the database, the left value is rounded. **money_prec*** is a number of decimal places for the currency of "money" type. When placing limit order, the price values shall be rounded, for stop orders (orderUpdated, pendingOrders, putStopLimitOrder, putStopOrder) – the triggerPrice value shall be rounded, and for the orderBook and orderBookUpdated methods – the precision value shall be rounded. **min_amount*** is a minimum amount for performing currency purchase transaction. When placing limit or market order, the amount value is rounded, and when receiving a list of orders from the database, the left value is rounded. NOTE. When executing marketList method, the user receives the maximum allowed number of values for all types of currency pairs for fee_prec, stock_prec, money_prec, and min_amount. If allowed number of characters exceeds the specified values, the query will be rejected.
Example of use	`query: '{marketList{name, stock, money, fee_prec, stock_prec, money_prec, min_amount}}',`

Example of response	`{ data:`
	`   { marketList:`
	`      [ { name: 'BTCUSDT',`
	`          stock: 'BTC',`
	`          money: 'USDT',`
	`          fee_prec: 4,`
	`          stock_prec: 4,`
	`          money_prec: 0,`
	`          min_amount: '0.0001' },`
	`        { name: 'ETHUSDT',`
	`          stock: 'ETH',`
	`          money: 'USDT',`
	`          fee_prec: 4,`
	`          stock_prec: 3,`
	`          money_prec: 0,`
	`          min_amount: '0.001' },`
	`        { name: 'BTCETH',`
	`          stock: 'BTC',`
	`          money: 'ETH',`
	`          fee_prec: 4,`
	`          stock_prec: 3,`
	`          money_prec: 0,`
	`          min_amount: '0.001' },`
	`        { name: 'ETHBTC',`
	`          stock: 'ETH',`
	`          money: 'BTC',`
	`          fee_prec: 4,`
	`          stock_prec: 3,`
	`          money_prec: 4,`
	`          min_amount: '0.001' } ] } }`

Table 13 - MarketToday Method (Daily market parameters for 24 hours)

Method name	**marketToday**
Method description	**Daily market parameters for 24 hours**
Protocol type	HTTP
Method type	Public method
Incoming parameters	**currency** is a name of currency (for example, BTC) or currency pair (for example, BTCUSDT) entering the market

Outgoing parameters	**market** is a market name; **stock** is a type of currency purchased; **money** is a monetary equivalent; **open** is a price for the currency pair selected at the time of Stock Exchange opening; **high** is a maximum price on the date of transaction for the currency pair selected; **low** is a minimum price on the date of transaction for the currency pair selected; **lastPrice** is a current price for the currency pair selected; **percentChange** is a change in price for the currency pair selected at the moment to the price for the currency pair selected at the time of Stock Exchange opening for the last 24 hours in percentage; **volume** is a volume of trades performed by currency types.
Example of use	 `query: '{marketToday(currency: "BTC"){market, stock, money, open, high, low, lastPrice, percentChange, volume}}'`

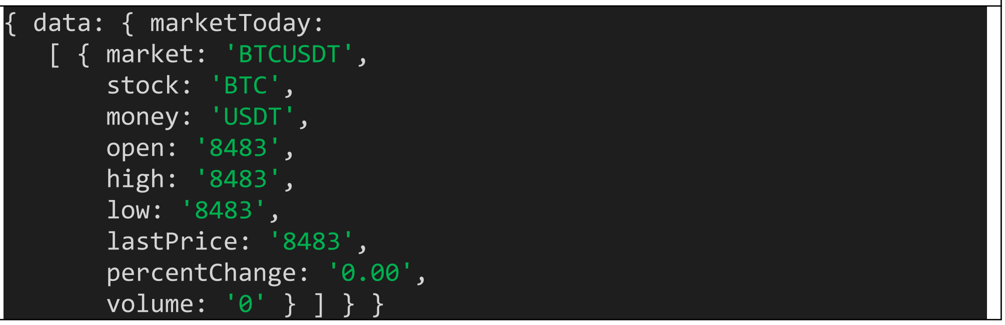
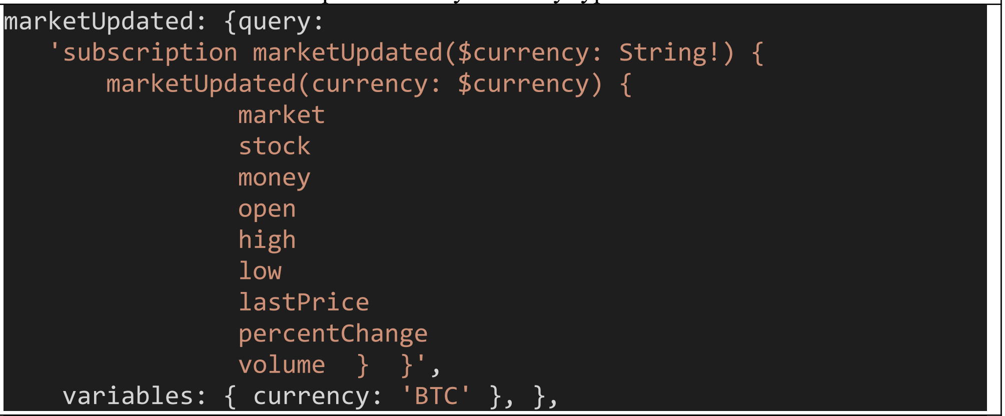
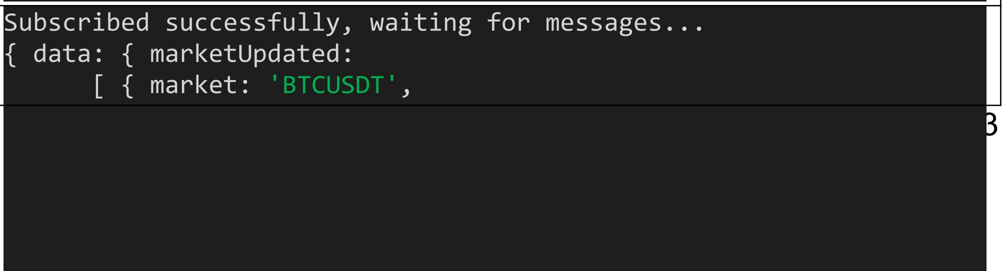
Example of response	
	`{ data: { marketToday:`
	`   [ { market: 'BTCUSDT',`
	`       stock: 'BTC',`
	`       money: 'USDT',`
	`       open: '8483',`
	`       high: '8483',`
	`       low: '8483',`
	`       lastPrice: '8483',`
	`       percentChange: '0.00',`
	`       volume: '0' } ] } }`

Table 14 - MarketUpdated Method (Subscription to the markets update in the currency selected)

Method name	**marketUpdated**
Method description	**Subscription to the markets update in the currency selected**
Protocol type	Websocket
Method type	Public method
Incoming parameters	**currency** is a name of currency (for example, BTC) or currency pair (for example, BTCUSDT) entering the market
Outgoing parameters	**market** is a market name; **stock** is a type of currency purchased; **money** is a monetary equivalent; **open** is a price for the currency pair selected at the time of Stock Exchange opening; **high** is a maximum price on the date of transaction for the currency pair selected; **low** is a minimum price on the date of transaction for the currency pair selected; **lastPrice** is a current price for the currency pair selected; **percentChange** is a change in price for the currency pair selected at the moment to the price for the currency pair selected at the time of Stock Exchange opening for the last 24 hours in percentage; **volume** is a volume of trades performed by currency types.

Example of use	
	`marketUpdated: {query:`
	`   'subscription marketUpdated($currency: String!) {`
	`       marketUpdated(currency: $currency) {`
	`                market`
	`                stock`
	`                money`
	`                open`
	`                high`
	`                low`
	`                lastPrice`
	`                percentChange`
	`                volume   }  }',`
	`    variables: { currency: 'BTC' }, },`

Example of response	
	`Subscribed successfully, waiting for messages...`
	`{ data: { marketUpdated:`
	`      [ { market: 'BTCUSDT',`

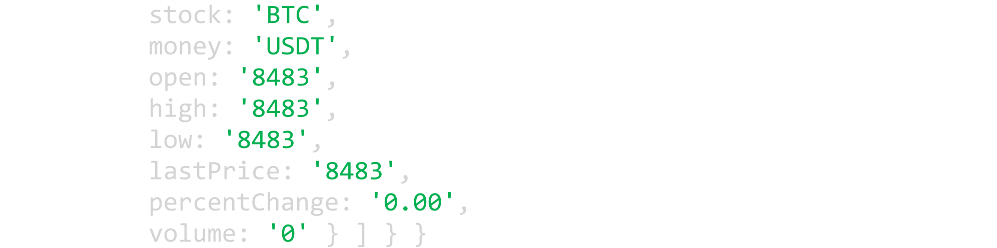
	 stock: 'BTC', money: 'USDT', open: '8483', high: '8483', low: '8483', lastPrice: '8483', percentChange: '0.00', volume: '0' }] } }

Table 15 - OrderBook Method (List of sale/purchase orders receipt)

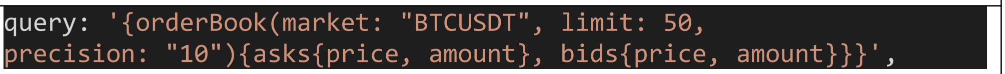
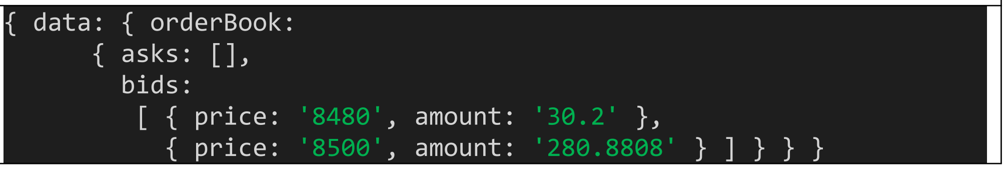
Method name	**orderBook**
Method description	**List of sale/purchase orders receipt**
Protocol type	HTTP
Method type	Public method
Incoming parameters	**market** is a market name; **limit** is a number of positions displayed (by default - 50); **precision*** is a step; offers close in price combination into a single position. NOTE. The precision value is rounded to the value specified in the query when receiving a list of sell/buy orders.
Outgoing parameters	**asks** is a list of sale orders (offer); **bids** is a list of purchase orders (demand); **price** is a price per unit of currency when buying / selling order; **amount** is a number of currency units displayed;
Example of use	 query: '{orderBook(market: "BTCUSDT", limit: 50, precision: "10"){asks{price, amount}, bids{price, amount}}}',
Example of response	 { data: { orderBook: { asks: [], bids: [{ price: '8480', amount: '30.2' }, { price: '8500', amount: '280.8808' }] } } }

Table 16 - OrderBookUpdated Method (Subscription to list update for the sale/purchase orders)

Method name	**orderBookUpdated**
Method description	**Subscription to the list update for the sale/purchase orders**
Protocol type	Websocket
Method type	Public method
Incoming parameters	**market** is a market name; **limit** is a number of positions displayed (by default - 50, but no more than 1000); **precision*** is a step; offers close in price combination into a single position; permissible values: "0" (default value - without orders grouping), "0.00000001", "0.0000001", "0.000001", "0.00001", "0.0001", "0.001", "0.01", "0.1". NOTE. Precision value is rounded to the value specified in the query when receiving a list of sale/purchase orders. If invalid value is set for any parameter value, the method will not be executed.
Outgoing parameters	**asks** is a list of sale orders (offer); **bids** is a list of purchase orders (demand); **price** is a price per unit of currency when buying / selling order; **amount** is a number of currency units displayed; **entirety** indicates integrity of queries book: - true - full list of requests receipt (when the query is initially executed); - false - updates to the list of requests receipt.
Example of use	orderBookUpdated: {query:

	'subscription orderBookUpdated($market: String!,
	$limit: Int! = 50, $precision: String! = "0") {
	orderBookUpdated(market: $market, limit: $limit,
	precision: $precision) {
	asks {price
	amount }
	bids {price
	amount }
	entirety } }',
	variables: { market: 'BTCUSDT', limit: 50,
	precision: '0.0001' }, },

Example of response	Subscribed successfully, waiting for messages... { data: { orderBookUpdated: { asks: [], bids: [{ price: '8480', amount: '30.2' }, { price: 8490', amount: '31.1505' },], entirety: true } } }

Table 17 - OrderHistory Method (User's order history)

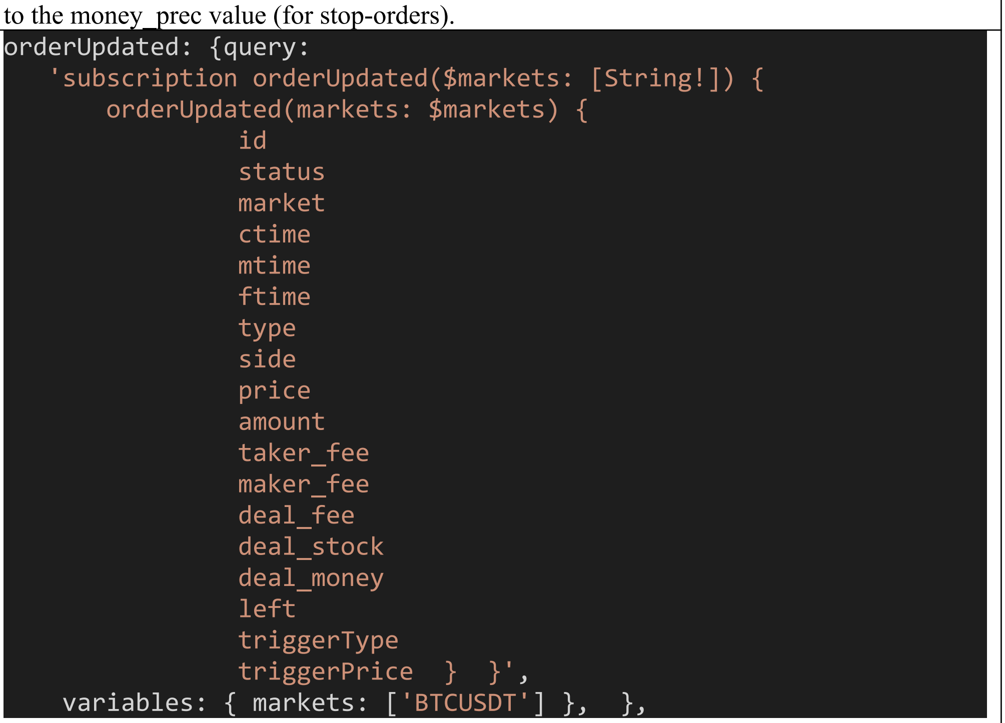
Method name	**orderHistory**
Method description	**User's order history**
Protocol type	HTTP
Method type	Private method
Incoming parameters	**market** is a market name; **startTime** is a start time of the day (in Unix Timestamp format); **endTime** is an end time of the day (in Unix Timestamp format); **offset** is a number of positions to skip from the beginning of the list (by default - 0); **limit** is a number of positions displayed (by default - 50); **side** is a type of orders transactions ("asks" - sell, "birds" - buy);
Outgoing parameters	**id** is an order ID; **status** is an order execution status ("pending" - pending execution, "updated" - partially executed, "finished" - fully executed, "canceled" - cancelled); **market** is a market name; **ctime** is a time when orders were created (in Unix Timestamp format); **mtime** is a time of the last order change (in Unix Timestamp format); **ftime** is an order closing time (in Unix Timestamp format); **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled. **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **price** is a price per unit of currency when buying / selling order; **amount** is a number of currency units displayed; **taker_fee** is a fee charged at the moment when the user purchases another user's offer; **maker_fee** is a fee charged at the moment when the user makes a new offer to purhase / sell order; **deal_money** is a volume of transaction in the orders sale; **deal_stock** is a transaction volume in the orders purchase; **deal_fee** is a fee on transaction; **left** is a not redeemed balance of order relative to the total amount.
Example of use	query: '{orderHistory(market: "BTCUSDT", startTime: 1579493168, endTime: 1579593700, offset: 0, limit: 50, side: "all") {id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_money, deal_stock, deal_fee, left}}',

Example of response	`{ data: { orderHistory:`
	`[ { id: 3909616,`
	`status: 'finished',`
	`market: 'BTCUSDT',`
	`ctime: '1579499237.188',`
	`mtime: '1579499237.18802',`
	`ftime: '1579499237.18802',`
	`type: 'Good-Till-Cancel',`
	`side: 'bids',`
	`price: '8483.0',`
	`amount: '0.1',`
	`taker_fee: '0.002',`
	`maker_fee: '0.002',`
	`deal_money: '8.641',`
	`deal_stock: '0.1',`
	`deal_fee: '0.0002',`
	`left: '0' } ] } }`

Table 18 - OrderUpdated Method (Subscription to change in active user orders status)

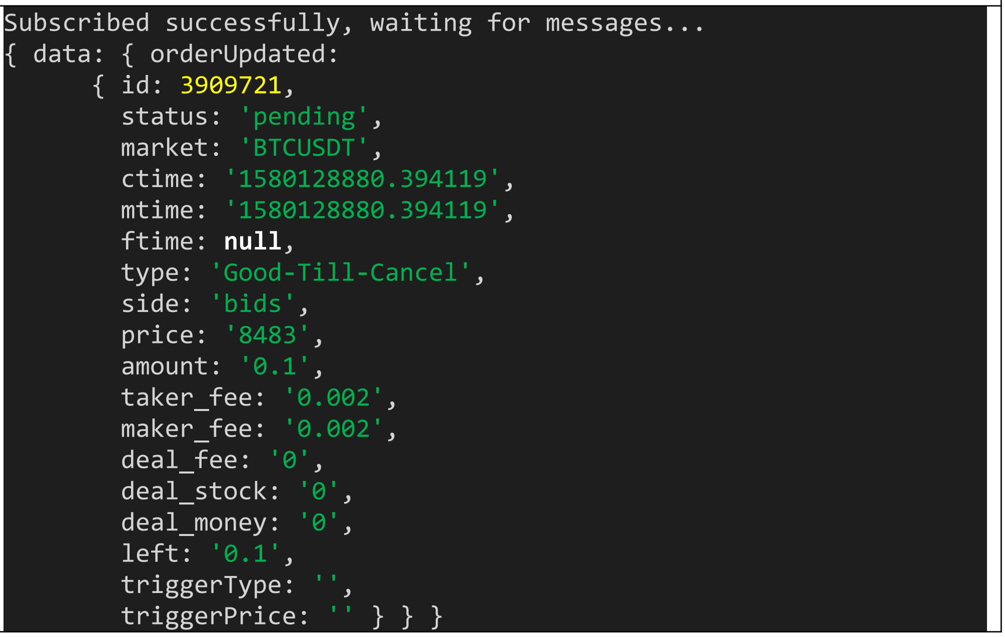
Method name	**orderUpdated**
Method description	**Subscription to change in active user orders status**
Protocol type	Websocket
Method type	Private method
Incoming parameters	**markets** is a list of markets the orders for which shall be tracked
Outgoing parameters	**id** is an order ID; **status** is an order execution status ("pending" - pending execution, "updated" - partially executed, "finished" - fully executed, "canceled" - cancelled); **market** is a market name; **ctime** is a time when orders were created (in Unix Timestamp format); **mtime** is a time of the last order change (in Unix Timestamp format); **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled. **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **price** is a price per unit of currency when buying / selling order; **amount** is a number of currency units displayed; **taker_fee** is a fee charged at the moment when the user purchases another user's offer; **maker_fee** is a fee charged at the moment when the user makes a new offer to buy / sell order; **deal_fee** is a fee on transaction; **deal_stock** is a transaction volume in the orders purchase; **deal_money** is a volume of transaction in the orders sale; **left*** is a not redeemed balance of order relative to the total amount. To select the order type "stop order" for the sale/purchase orders to be placed, tick "Stop/Take profit" field, specify transaction type - "ask/bid" and specify maximum permissible value in the "Stop-price" field. Parameters assigned for the "stop order": **parentId** is an ID for the parent order placed when receiving counter order under previously set conditions. In all other cases, the value is "null". **ftime** is an order closing time (in Unix Timestamp format); In all other cases, the value is "null". **triggerType** is a flag for the order type "stop order" selection (" "- flag is not set, "ask" - "stop order" of "ask" type, "bid" - "stop order" of "bid" type); **triggerPrice**** is a maximum permissible value in the "Stop price" field. In all other cases, the value is "".

<table>
<tr><td></td><td>NOTE.
Maximum permissible number of values for the currency pair selected for stock_prec, money_prec is defined in the marketList method. If allowed number of characters exceeds the specified values, the query will be rejected.
* left. The parameter value is rounded by deleting additional characters according to the stock_prec value (when receiving a list of orders from the database);
** triggerPrice. The parameter value is rounded by deleting additional characters according to the money_prec value (for stop-orders).</td></tr>
<tr><td><b>Example of use</b></td><td>



```
orderUpdated: {query:
   'subscription orderUpdated($markets: [String!]) {
       orderUpdated(markets: $markets) {
                id
                status
                market
                ctime
                mtime
                ftime
                type
                side
                price
                amount
                taker_fee
                maker_fee
                deal_fee
                deal_stock
                deal_money
                left
                triggerType
                triggerPrice  }  }',
    variables: { markets: ['BTCUSDT'] },  },
```

</td></tr>
<tr><td><b>Example of response</b></td><td>



```
Subscribed successfully, waiting for messages...
{ data: { orderUpdated:
      { id: 3909721,
        status: 'pending',
        market: 'BTCUSDT',
        ctime: '1580128880.394119',
        mtime: '1580128880.394119',
        ftime: null,
        type: 'Good-Till-Cancel',
        side: 'bids',
        price: '8483',
        amount: '0.1',
        taker_fee: '0.002',
        maker_fee: '0.002',
        deal_fee: '0',
        deal_stock: '0',
        deal_money: '0',
        left: '0.1',
        triggerType: '',
        triggerPrice: '' } } }
```

</td></tr>
</table>

Table 19 - PendingOrders Method (User's order opening)

Method name	**pendingOrders**
Method description	**User's order opening**
Protocol type	HTTP
Method type	Private method
Incoming parameters	**market** is a market name; **offset** is a number of positions to skip from the beginning of the list (by default - 0); **limit** is a number of positions displayed (by default - 100).

Outgoing parameters	**id** is an order ID; **status** is an order execution status ("pending" - pending execution, "updated" - partially executed, "finished" - fully executed, "canceled" - cancelled); **market** is a market name; **ctime** is a time when orders were created (in Unix Timestamp format); **mtime** is a time of the last order change (in Unix Timestamp format); **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, and executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled. **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **price** is a price per unit of currency when buying / selling order; **amount** is a number of currency units displayed; **taker_fee** is a fee charged at the moment when the user purchases another user's offer; **maker_fee** is a fee charged at the moment when the user makes a new offer to buy / sell order; **deal_fee** is a fee on transaction; **deal_stock** is a transaction volume in the orders purchase; **deal_money** is a volume of transaction in the orders sale; **left*** is a not redeemed balance of order relative to the total amount. To select the order type "stop order" for the sale/purchase orders to be placed, tick "Stop/Take profit" field, specify transaction type - "ask/bid" and specify maximum permissible value in the "Stop-price" field. Parameters assigned for the "stop order": **parentId** is an ID for the parent order placed when receiving counter order under previously set conditions. In all other cases, the value is "null". **ftime** is an order closing time (in Unix Timestamp format); In all other cases, the value is "null". **triggerType** is a flag for the order type "stop order" selection (" "- flag is not set, "ask" - "stop order" of "ask" type, "bid" - "stop order" of "bid" type); **triggerPrice**** is a maximum permissible value in the "Stop price" field. In all other cases, the value is "". NOTE. The maximum permissible number of values for the currency pair selected for stock_prec, money_prec is defined in the marketList method. If allowed number of characters exceeds the specified values, the query will be rejected. * left. The parameter value is rounded by deleting additional characters according to the stock_prec value (when receiving a list of orders from the database); ** triggerPrice. The parameter value is rounded by deleting additional characters according to the money_prec value (for stop-orders).

Example of use	`query: '{pendingOrders(market: "BTCUSDT", offset: 0, limit: 100){id`
	`,`
	`parentId, status, market, ctime, mtime, ftime, type, side, price,`
	`amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_money,`
	`left, triggerType, triggerPrice}}',`

Example of response	`{ data: { pendingOrders:`
	`      [ { id: 3909653,`
	`          parentId: null,`
	`          status: 'pending',`
	`          market: 'BTCUSDT',`
	`          ctime: '1579607589.58058',`
	`          mtime: '1579607589.58058',`
	`          ftime: null,`
	`          type: 'Good-Till-Cancel',`
	`          side: 'asks',`
	`          price: '8483',`
	`          amount: '1',`
	`          taker_fee: '0.002',`

	maker_fee: '0.002', deal_fee: '0', deal_stock: '0', deal_money: '0', left: '1', triggerType: 'ask', triggerPrice: '8480' }] } }

Table 20 - PendingOrderDetail Method (Pending user's order)

Method name	**pendingOrderDetail**
Method description	**Pending user's order**
Protocol type	HTTP
Method type	Private method
Incoming parameters	**market** is a market name; **orderId** is an ordinal number of order.
Outgoing parameters	**id** is an order ID; **status** is an order execution status ("pending" - pending execution, "updated" - partially executed, "finished" - fully executed, "canceled" - cancelled); **market** is a market name; **ctime** is a time when orders were created (in Unix Timestamp format); **mtime** is a time of the last order change (in Unix Timestamp format); **ftime** is an order closing time (in Unix Timestamp format); The time is defined only for the "stop-order", otherwise the value is "null". **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled. **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **price** is a price per unit of currency when buying / selling order; **amount** is a number of currency units displayed; **taker_fee** is a fee charged at the moment when the user purchases another user's offer; **maker_fee** is a fee charged at the moment when the user makes a new offer to buy / sell order; **deal_money** is a volume of transaction in the orders sale; **deal_stock** is a transaction volume in the orders purchase; **deal_fee** is a fee on transaction; **left** is a not redeemed balance of order relative to the total amount.
Example of use	query: '{pendingOrderDetail(market: "BTCUSDT", orderId: 980026){id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_money, deal_stock, deal_fee, left}}',
Example of response	{ data: { pendingOrderDetail: { id: 3909654, status: 'pending', market: 'BTCUSDT', ctime: '1579607607.14803', mtime: '1579607607.14803', ftime: null, type: 'Good-Till-Cancel', side: 'asks', price: '8483', amount: '1', taker_fee: '0.002', maker_fee: '0.002', deal_money: '0', deal_stock: '0',

	`deal_fee: '0',`
	`left: '1' } } }`

Table 21 - PendingSummary Method (Number of user`s orders by markets)

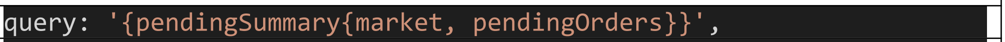
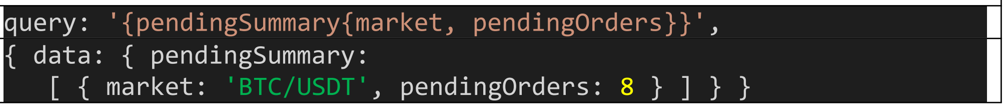
Method name	**pendingSummary**
Method description	**Number of user`s orders by markets**
Protocol type	HTTP
Method type	Private method
Incoming parameters	—
Outgoing parameters	**market** is a market name; **pendingOrders** is a number of user`s orders by markets.
Example of use	 `query: '{pendingSummary{market, pendingOrders}}',`
Example of response	 `{ data: { pendingSummary:` `[ { market: 'BTC/USDT', pendingOrders: 8 } ] } }`

Table 22 - PutLimitOrder Method (Order of the "limit" type placing)

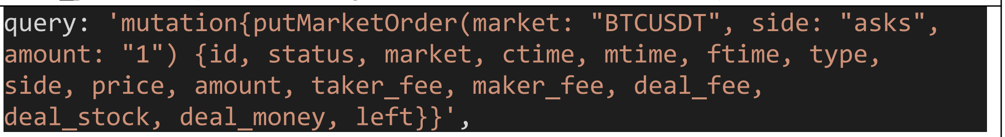
Method name	**putLimitOrder**
Method description	**Order of the "limit" type placing that indicates the maximum price to broker, at which the user is ready to buy or the minimum price at which the user is ready to sell the currency**
Protocol type	HTTP
Method type	Private method
Incoming parameters	**market** is a market name; **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **amount** is a number of currency units displayed; **price** is a price per unit of currency when buying / selling order; **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled.
Outgoing parameters	**id** is an order ID; **status** is an order execution status ("pending" - pending execution, "updated" - partially executed, "finished" - fully executed, "canceled" - cancelled); **market** is a market name; **ctime** is a time when orders were created (in Unix Timestamp format); **mtime** is a time of the last order change (in Unix Timestamp format); **ftime** is an order closing time (in Unix Timestamp format); **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, and executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled. **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **price*** is a price per unit of currency when buying / selling order; **amount**** is a number of currency units displayed; **taker_fee** is a fee charged at the moment when the user purchases another user's offer; **maker_fee** is a fee charged at the moment when the user makes a new offer to buy / sell order; **deal_fee** is a fee on transaction; **deal_stock** is a transaction volume in the orders purchase; **deal_money** is a volume of transaction in the orders sale; **left***** is a not redeemed balance of order relative to the total amount. NOTE.

<table>
<tr><td></td><td>The maximum permissible number of values for the currency pair selected for stock_prec, money_prec is defined in the marketList method. If allowed number of characters exceeds the specified values, the query will be rejected.
* price. The parameter value is rounded by deleting additional characters according to the money_prec value (when placing limit order).
** amount. The parameter value is rounded by deleting additional characters according to the stock_prec value (when placing limit order);
*** left. The parameter value is rounded by deleting additional characters according to the stock_prec value (when receiving a list of orders from the database).</td></tr>
<tr><td><b>Example of use</b></td><td><pre>query: 'mutation{putLimitOrder(market: "BTCUSDT", side: "asks",
amount: "1", price: "10", type: "Good-Till-Cancel") {id, status,
market, ctime, mtime, ftime, type,   side, price, amount,
taker_fee, maker_fee, deal_fee, deal_stock, deal_money,
left}}',</pre></td></tr>
<tr><td><b>Example of response</b></td><td><pre>{ data: { putLimitOrder:
      { id: 3909722,
        status: 'finished',
        market: 'BTCUSDT',
        ctime: '1580135269.026323',
        mtime: '1580135269.026336',
        ftime: '1580135269.026336',
        type: 'Good-Till-Cancel',
        side: 'asks',
        price: '8483',
        amount: '1',
        taker_fee: '0.002',
        maker_fee: '0.002',
        deal_fee: '0.02',
        deal_stock: '1',
        deal_money: '10',
        left: '0' } } }</pre></td></tr>
</table>

Table 23 - PutMarketOrder Method (Order of the "market" type placing)

Method name	**putMarketOrder**
Method description	**Order of the "market" type placing executed immediately when the limit order with the best current price is found**
Protocol type	HTTP
Method type	Private method
Incoming parameters	**market** is a market name; **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **amount** is a number of currency units displayed;
Outgoing parameters	**id** is an order ID; **status** is an order execution status ("pending" - pending execution, "updated" - partially executed, "finished" - fully executed, "canceled" - cancelled); **market** is a market name; **ctime** is a time when orders were created (in Unix Timestamp format); **mtime** is a time of the last order change (in Unix Timestamp format); **ftime** is an order closing time (in Unix Timestamp format); **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, and executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled. **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **price*** is a price per unit of currency when purchasing / selling order; **amount**** is a number of currency units displayed; **taker_fee** is a fee charged at the moment when the user purchases another user's offer; **maker_fee** is a fee charged at the moment when the user makes a new offer to buy / sell

	order; **deal_fee** is a fee on transaction; **deal_stock** is a transaction volume in the orders purchase; **deal_money** is a volume of transaction in the orders sale; **left***** is a not redeemed balance of order relative to the total amount. NOTE. The maximum permissible number of values for the currency pair selected for stock_prec, money_prec is defined in the marketList method. If allowed number of characters exceeds the specified values, the query will be rejected. * price. The parameter value is rounded by deleting additional characters according to the money_prec value (when placing market order). ** amount. The parameter value is rounded by deleting additional characters according to the stock_prec value (when placing market order); *** left. The parameter value is rounded by deleting additional characters according to the stock_prec value (when receiving a list of orders from the database).

Example of use	
	`query: 'mutation{putMarketOrder(market: "BTCUSDT", side: "asks",`
	`amount: "1") {id, status, market, ctime, mtime, ftime, type,`
	`side, price, amount, taker_fee, maker_fee, deal_fee,`
	`deal_stock, deal_money, left}}',`

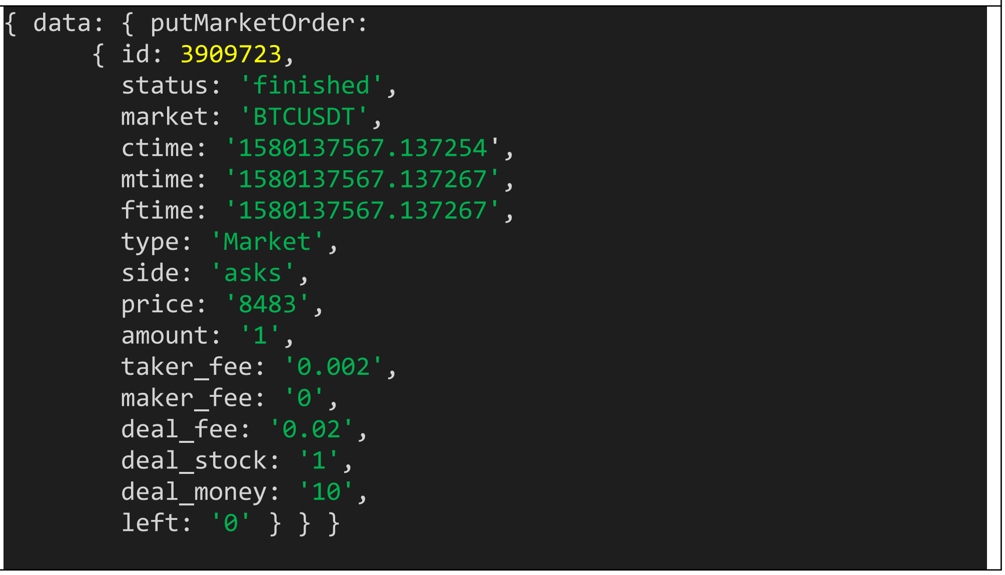
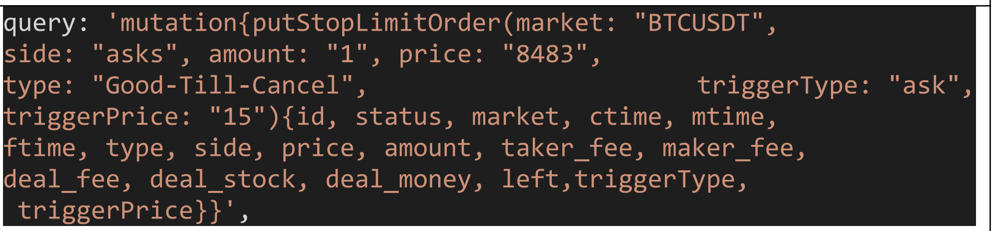
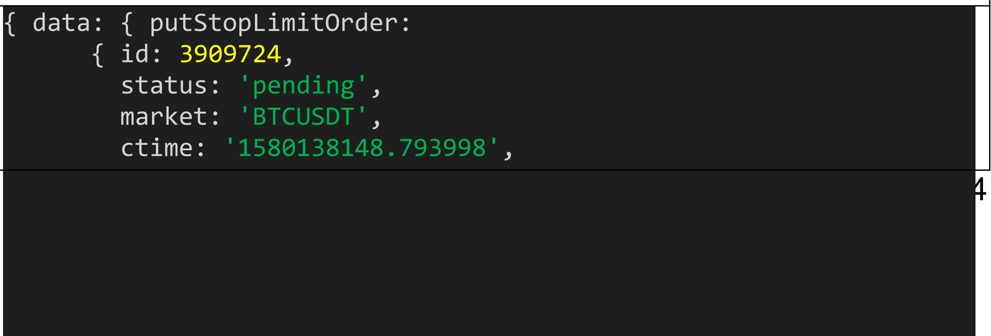
Example of response	
	`{ data: { putMarketOrder:`
	`      { id: 3909723,`
	`        status: 'finished',`
	`        market: 'BTCUSDT',`
	`        ctime: '1580137567.137254',`
	`        mtime: '1580137567.137267',`
	`        ftime: '1580137567.137267',`
	`        type: 'Market',`
	`        side: 'asks',`
	`        price: '8483',`
	`        amount: '1',`
	`        taker_fee: '0.002',`
	`        maker_fee: '0',`
	`        deal_fee: '0.02',`
	`        deal_stock: '1',`
	`        deal_money: '10',`
	`        left: '0' } } }`

Table 24 - PutStopLimitOrder Method (Stop-order of the "limit" type placing)

Method name	**putStopLimitOrder**
Method description	**Stop-order of the "limit" type placing indicating the purchase price per unit of currency. The order is placed in the list of orders when the market price (min_ask or max_bid) achieved the stop-price value**
Protocol type	HTTP
Method type	Private method
Incoming parameters	**market** is a market name; **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **amount** is a number of currency units displayed; **price** is a price per unit of currency when buying / selling order; **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, and executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled. **triggerType** is a flag for the order type "stop order" selection (" "- flag is not set, "ask" - "stop order" of "ask" type, "bid" - "stop order" of "bid" type); **triggerPrice** is a maximum permissible value in the "Stop- price" field.
Outgoing	**id** is an order ID;

parameters	**status** is an order execution status ("pending" - pending execution, "updated" - partially executed, "finished" - fully executed, "canceled" - cancelled); **market** is a market name; **ctime** is a time when orders were created (in Unix Timestamp format); **mtime** is a time of the last order change (in Unix Timestamp format); **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, and executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled. **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **price*** is a price per unit of currency when buying / selling order; **amount**** is a number of currency units displayed; **taker_fee** is a fee charged at the moment when the user purchases another user's offer; **maker_fee** is a fee charged at the moment when the user makes a new offer to buy / sell order; **deal_fee** is a fee on transaction; **deal_stock** is a transaction volume in the orders purchase; **deal_money** is a volume of transaction in the orders sale; **left***** is a not redeemed balance of order relative to the total amount. To select the order type "stop order" for the sale/purchase orders to be placed, tick "Stop/Take profit" field, specify transaction type - "ask/bid" and specify maximum permissible value in the "Stop-price" field. Parameters assigned for the "stop order": **ftime** - order closing time (in Unix Timestamp format); In all other cases, the value is "null". **triggerType** is a flag for the order type "stop order" selection (" "- flag is not set, "ask" - "stop order" of "ask" type, "bid" - "stop order" of "bid" type); **triggerPrice****** is a maximum permissible value in the "Stop price" field. In all other cases, the value is "". NOTE. The maximum permissible number of values for the currency pair selected for stock_prec, money_prec is defined in the marketList method. If allowed number of characters exceeds the specified values, the query will be rejected. * price. The parameter value is rounded by deleting additional characters according to the money_prec value (when placing limit order). ** amount. The parameter value is rounded by deleting additional characters according to the stock_prec value (when placing limit order); *** left. The parameter value is rounded by deleting additional characters according to the stock_prec value (when receiving a list of orders from the database). **** triggerPrice. The parameter value is rounded by deleting additional characters according to the money_prec value (for stop-orders).

Example of use	
	`query: 'mutation{putStopLimitOrder(market: "BTCUSDT",`
	`side: "asks", amount: "1", price: "8483",`
	`type: "Good-Till-Cancel",                     triggerType: "ask",`
	`triggerPrice: "15"){id, status, market, ctime, mtime,`
	`ftime, type, side, price, amount, taker_fee, maker_fee,`
	`deal_fee, deal_stock, deal_money, left,triggerType,`
	` triggerPrice}}',`

Example of response	
	`{ data: { putStopLimitOrder:`
	`      { id: 3909724,`
	`        status: 'pending',`
	`        market: 'BTCUSDT',`
	`        ctime: '1580138148.793998',`

	mtime: '1580138148.793998', ftime: , type: 'Good-Till-Cancel', side: 'asks', price: '8483', amount: '1', taker_fee: '0.002', maker_fee: '0.002', deal_fee: '0', deal_stock: '0', deal_money: '0', left: '1', triggerType: 'ask', triggerPrice: '15' } } }

Table 25 - PutStopOrder Method (Stop-order of the "market" type placing)

Method name	**putStopOrder**
Method description	**Stop-order of the "market" type placing indicating the amount of currency purchased. The order is placed in the list of orders when the market price (min_ask or max_bid) achieved the stop-price value**
Protocol type	HTTP
Method type	Private method
Incoming parameters	**market** is a market name; **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **amount** is a number of currency units displayed; **triggerType** is a flag for the order type "stop order" selection (" "- flag is not set, "ask" - "stop order" of "ask" type, "bid" - "stop order" of "bid" type); **triggerPrice** is a maximum permissible value in the "Stop-price" field.
Outgoing parameters	**id** is an order ID; **status** is an order execution status ("pending" - pending execution, "updated" - partially executed, "finished" - fully executed, "canceled" - cancelled); **market** is a market name; **ctime** is a time when orders were created (in Unix Timestamp format); **mtime** is a time of the last order change (in Unix Timestamp format); **type** is an order type: - "Market" - market type; - "Good-Till-Cancel" - limited, active until manually cancelled; - "Immediate-or-Cancel" - limited, executed in full or in part at the moment of creation, executed order part is cancelled; - "Fill-or-Kill" - should be executed in full at the price set (or better) or cancelled. **side** is a type of orders transactions ("asks" - sell, "birds" - buy); **price*** is a price per unit of currency when buying / selling order; **amount**** is a number of currency units displayed; **taker_fee** is a fee charged at the moment when the user purchases another user's offer; **maker_fee** is a fee charged at the moment when the user makes a new offer to buy / sell order; **deal_fee** is a fee on transaction; **deal_stock** is a transaction volume in the orders purchase; **deal_money** is a volume of transaction in the orders sale; **left***** is a not redeemed balance of order relative to the total amount. To select the order type "stop order" for the sale/purchase orders to be placed, tick "Stop/Take profit" field, specify transaction type - "ask/bid" and specify maximum permissible value in the "Stop-price" field. Parameters assigned for the "stop order": **ftime** is an order closing time (in Unix Timestamp format); In all other cases, the value is "null". **triggerType** is a flag for the order type "stop order" selection (" "- flag is not set, "ask" - "stop order" of "ask" type, "bid" - "stop order" of "bid" type); **triggerPrice****** is a maximum permissible value in the "Stop price" field. In all other

<table>
<tr><td></td><td>cases, the value is "".

NOTE.
The maximum permissible number of values for the currency pair selected for stock_prec, money_prec is defined in the marketList method. If allowed number of characters exceeds the specified values, the query will be rejected.
* price. The parameter value is rounded by deleting additional characters according to the money_prec value (when placing market order).
** amount. The parameter value is rounded by deleting additional characters according to the stock_prec value (when placing market order);
*** left. The parameter value is rounded by deleting additional characters according to the stock_prec value (when receiving a list of orders from the database).
**** triggerPrice. The parameter value is rounded by deleting additional characters according to the money_prec value (for stop-orders).</td></tr>
<tr><td><b>Example of use</b></td><td><pre>
  query: 'mutation{putStopOrder(market: "BTCUSDT", side: "asks",
 amount: "1", triggerType: "ask", triggerPrice: "8480") {id,
 status, market, ctime, mtime, ftime, type, side, price,
 amount, taker_fee, maker_fee, deal_fee, deal_stock,
 deal_money, left, triggerType, triggerPrice}}'
</pre></td></tr>
<tr><td><b>Example of response</b></td><td><pre>
{ data: { putStopOrder:
      { id: 3909725,
        status: 'pending',
        market: 'BTCUSDT',
        ctime: '1580138846.509452',
        mtime: '1580138846.509452',
        ftime: null,
        type: 'Market',
        side: 'asks',
        price: '0',
        amount: '1',
        taker_fee: '0.002',
        maker_fee: '0',
        deal_fee: '0',
        deal_stock: '0',
        deal_money: '0',
        left: '1',
        triggerType: 'ask',
        triggerPrice: '8480' } } }
</pre></td></tr>
</table>

Table 26 - StateUpdated Method (Subscription to the information update for the market selected)

Method name	**stateUpdated**
Method description	**Subscription to the information update for the market selected**
Protocol type	Websocket
Method type	Public method
Incoming parameters	**market** is a market name
Outgoing parameters	**open** is a price for the currency pair selected at the time of Stock Exchange opening; **market** is a market name; **stock** is a type of currency purchased; **money** is a monetary equivalent; **high** is a maximum price on the date of transaction for the currency pair selected; **low** is a minimum price on the date of transaction for the currency pair selected; **lastPrice** is a current price for the currency pair selected; **percentChange** is a change in price for the currency pair selected at the moment to the price for the currency pair selected at the time of Stock Exchange opening for the last 24 hours in percentage; **volume** is a volume of trades performed by currency types.

<table>
<tr><td></td><td></td></tr>
<tr><td><b>Example of use</b></td><td><pre>stateUpdated: {query:
   'subscription stateUpdated($market: String!) {
       stateUpdated(market: $market) {
                open
                market
                stock
                money
                high
                low
                lastPrice
                percentChange
                volume } }',
    variables: { market: 'BTCUSDT' }, },</pre></td></tr>
<tr><td><b>Example of response</b></td><td><pre>Subscribed successfully, waiting for messages...
{ data: { stateUpdated:
      { open: '8480,
        market: 'BTCUSDT',
        stock: 'BTC',
        money: 'USDT',
        high: '8483',
        low: '8480',
        lastPrice: '8483',
        percentChange: '0.03',
        volume: '7.5326' } } }</pre></td></tr>
</table>

Table 27 - TradeHistory Method (History of transactions receipt)

<table>
<tr><td><b>Method name</b></td><td><b>tradeHistory</b></td></tr>
<tr><td><b>Method description</b></td><td><b>History of transactions receipt</b></td></tr>
<tr><td><b>Protocol type</b></td><td>HTTP</td></tr>
<tr><td><b>Method type</b></td><td>Public method</td></tr>
<tr><td><b>Incoming parameters</b></td><td><b>market</b> is a market name;
<b>limit</b> is a number of positions displayed (by default - 10);
<b>offset</b> is a number of positions to skip from the beginning of the list (by default - 100).</td></tr>
<tr><td><b>Outgoing parameters</b></td><td><b>id</b> is an order ID;
<b>time</b> is a query completion time (in Unix Timestamp format);
<b>price</b> is a price per unit of currency when buying / selling order;
<b>amount</b> is a number of currency units displayed;
<b>type</b> is a type of transaction ("sell", "buy").</td></tr>
<tr><td><b>Example of use</b></td><td><pre>query: '{tradeHistory(market: "BTCUSDT", limit: 10, offset: 100)
{id, time, price, amount, type}}',</pre></td></tr>
<tr><td><b>Example of response</b></td><td><pre>{ data: { tradeHistory:
      [{ id: '200649',
          time: '1580137567.137267',
          price: '8483',
          amount: '1',
          type: 'sell' }] } }</pre></td></tr>
</table>

Table 28 - WalletCommissions Method (Amount of commission for the types of currencies debited from the user when withdrawing funds from Stock Exchange)

<table>
<tr><td><b>Method name</b></td><td><b>walletCommissions</b></td></tr>
<tr><td><b>Method description</b></td><td><b>Amount of commission for the types of currencies debited from the user when withdrawing funds from the Stock Exchange</b></td></tr>
<tr><td><b>Protocol type</b></td><td>HTTP</td></tr>
<tr><td><b>Method type</b></td><td>Public method</td></tr>
<tr><td><b>Incoming parameters</b></td><td>—</td></tr>
<tr><td><b>Outgoing</b></td><td><b>name</b> is a name of currency used on the Stock Exchange (cryptocurrencies and fiat</td></tr>
</table>

parameters	money); **commission** is an amount of commission for the types of currencies debited from the user when withdrawing funds from the Stock Exchange.
Example of use	`query: '{walletCommissions {name, commission}}',`

Example of response	`{ data: { walletCommissions:`
	`      [ { name: 'ETH', commission: '0.000861' },`
	`        { name: 'BTC', commission: '0.00000167' },`
	`        { name: 'USDT', commission: '0' } ] } }`

Table 29 - WalletQuery Method (Wallet address receipt for the cryptocurrencies entry)

Method name	**walletQuery**
Method description	**Wallet address receipt for the cryptocurrencies entry**
Protocol type	HTTP
Method type	Private method
Incoming parameters	**currency** is a currency name entering the market;
Outgoing parameters	**address** is an address of the cryptocurrency wallet; **message** is a secret seed-phrase required to create a backup copy or restore the wallet. It is used to create certain types of cryptocurrencies, such as Monero.
Example of use	`query: '{walletQuery(currency: "BTC"){address, message}}',`

Example of response	`{ data: { walletQuery:`
	`      { address: 'Address of the cryptocurrency wallet',`
	`        message: null } } }`

2.3 Output data

Section "Query Structure" provides general information on the output data setup (for more details, see section 2.1, paragraph "Information output to the web-console") and the structure of the information output.

The response output format is sent as a message to the web-console with parameters according to the query format (possible types of information output to console):

1) Notification output about method execution result.

```
console.info(body.data, body.errors, error)
```

Format of response when method is successfully executed:

```
{ orderBook: { asks: [], bids: [ [Object], [Object] ] } } undefined null
```

Format of response when syntax error occurs:

```
query: '{orderBook(market: 'BTCUSDT', limit: 50, precision: "10"){asks{price,
                            ^^^^^^
amount}, bids{price, amount}}}',

SyntaxError: Unexpected identifier
    at new Script (vm.js:85:7)
    at createScript (vm.js:266:10)
    at Object.runInThisContext (vm.js:314:10)
    at Module._compile (internal/modules/cjs/loader.js:698:28)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:749:10)
    at Module.load (internal/modules/cjs/loader.js:630:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:570:12)
    at Function.Module._load (internal/modules/cjs/loader.js:562:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:801:12)
    at internal/main/run_main_module.js:21:11
```

2) Structure output of the object passed with query data disclosure up to the specified depth level:

- HTTP method:

```
console.dir(response.body,{depth:10})
```

- Websocket method:

```
console.dir(eventData,{depth:10})
```

Response format for the HTTP method:

```
{ data: { orderBook:
      { asks: [],
        bids:
         [ { price: '8480', amount: '30.2' },
           { price: '8500', amount: '280.8808' } ] } } }
```

NOTE:

To correctly display the values received specify maximum depth level. Example of invalid level of nesting:

```
{ data: { orderBook: [Object] } }
```

3) Server settings transmission when executing query.

```
console.log(response)
```

Example shows small fragment, since the full text of response includes several pages.

```
IncomingMessage {
  _readableState:
   ReadableState {
     objectMode: false,
     highWaterMark: 16384,
     buffer: BufferList { head: null, tail: null, length: 0 },
     length: 0,
     pipes: null,
     pipesCount: 0,
     flowing: true,
     ended: true,
     endEmitted: true,
     reading: false,
     sync: false,
     needReadable: false,
     emittedReadable: false,
     readableListening: false,
     resumeScheduled: false,
     paused: false,
     emitClose: true,
     autoDestroy: false,
     destroyed: false,
     defaultEncoding: 'utf8',
     awaitDrain: 0,
     readingMore: false,
     decoder: null,
     encoding: null },
  readable: false,
  _events:
```

4) List of parameters passed in query and query text receipt.

```
console.log(options)
```

The response contains settings transmitted to the server (url address, Unix time transfer format in "nonce" string, public apiKey, signature, and query text).

```
{ contentType: 'application/json',
  url: 'https://bizonex.com/api/v1',
  headers:
   { nonce: 1561381313,
     apiKey:
      'Enter public key Token',
     signature:
      'Enter private key Token' },
  body:
   { query:
      '{orderBook(market: "BTCUSDT", limit: 50, precision: "10"){asks{price, amount},
bids{price, amount}}}' },
  json: true }
```

5) Informational message for the Websocket methods that subscription to the method has been made, and the log is waiting for the event.

```
console.log(`Subscribed successfully, waiting for messages...`)
```

Response format:

```
Sibscribed successfully, waiting for messages...
```

2.4 Error description

This section lists any types of errors occurring in Stock Exchange API methods execution (Table 30) displayed as messages in the web-console:

- Business-logic errors from the Next log - Table 30.

Errors of this type are associated with the rights, available tools, and allowed actions for a specific user.

- System errors from the Error log - Table 31.

This type of error occurs if the server is unavailable, system time inconsistency on the user's PC and Stock Exchange server, or if the current task cannot be processed or information transmitted between the client and the server.

Table 30 – Description of business-logic errors occurring in API methods execution

No.	Characteristics of errors	Examples of business-logic errors
1	*Name of the method/methods*	**cancelOrder**
	Cause of error	There is no active order with the orderId number specified.

	Query example	`query:`
		`'mutation{cancelOrder(market: "BTCUSDT", orderId: 12345) {id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_money, left} }',`

	Query result	`{ errors: [ { message: 'An error occurred while canceling pending order 12345.There is no active order with provided ID',`
		`path: [ 'cancelOrder' ],`
		`extensions: { title: 'Order not found',`
		`code: 'ORDER_NOT_FOUND',`
		`orderId: 12345 } } ],`
		`data: }`

2	*Name of the method/methods*	**cancelOrder**
	Cause of error	Invalid parameter record format. In example the orderId parameter is assigned with the value type "Integer" instead of "String".
	Query example	`query:`

		'mutation{cancelOrder(market: "BTCUSDT", orderId: 12345){id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_money, left} }',
	Query result	{ errors: [{ message: 'Expected type String!, found BTCUSDT.', extensions: { code: 'GRAPHQL_VALIDATION_FAILED' } }] }
3	*Name of the method/methods*	**putLimitOrder, putStopLimitOrder**
	Cause of error	Allowed number of decimal places for the currency types is exceeded. If one of the currencies in the pair is fiat money, the number of decimal places should not exceed two characters. for more details, see currencyList method.
	Query example	query: 'mutation{putLimitOrder(market:"BTCUSDT", side: "asks", amount: "1",price: "0.0001", type: "Good-Till-Cancel") {id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_money, left}}',
	Query result	{ errors: [{ message: 'Internal server error occurred', path: ['putLimitOrder'], extensions: { title: 'Internal server error', code: 'INTERNAL_SERVER_ERROR' } }], data: }
4	*Name of the method/methods*	**putLimitOrder, putStopLimitOrder, putMarketOrder**
	Cause of error	The amount of currency placed in order is less than the allowed amount (the value of amount parameter is too small). The error is displayed when changing the accuracy of stock_prec, money_prec, or min_amount parameters.
	Query example	query: 'mutation{putLimitOrder(market: "BTCUSDT", side: "asks" , amount: "0.000001", price: "8483", type: "Good-Till-Cancel") {id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_fee, deal_stock deal_money, left}}',
	Query result	{ errors: [{ message: 'The provided amount is too small to execute an order', path: ['putLimitOrder'], extensions: { title: 'Too small amount', code: 'TOO_SMALL_AMOUNT', minAmount: '0.00001' } }], data: }
5	*Name of the method/methods*	**putLimitOrder, putMarketOrder, putStopLimitOrder, putStopOrder**
	Cause of error	Invalid value for amount parameter.
	Query example	query: 'mutation{putLimitOrder(market: "BTCUSDT", side: "asks", amount: "0.00000000001", price: "1", type: "Good-Till- Cancel"){id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_money, left}}',
	Query result	{ errors: [{ message: 'Internal server error occurred', path: ['putLimitOrder'], extensions: { title: 'Internal server error', code: 'INTERNAL_SERVER_ERROR' } }], data: }

6	*Name of the method/methods*	**putLimitOrder, putStopLimitOrder, putMarketOrder**
	Cause of error	The available balance is not sufficient by currency type to make transaction.

	Query example	`query: 'mutation{putLimitOrder(market: "BTCUSDT", side: "asks"`
		`,`
		` amount: "1000000000000000000000", price: "1", type: "Good-`
		`Till-Cancel") {id, status, market, ctime, mtime, ftime,`
		` type, side, price, amount, taker_fee, maker_fee, deal_fee,`
		` deal_stock,deal_money, left}}',`

	Query result	`{ errors: [ { message: 'The available balance is not enough to`
		`execute an order',`
		`    path: [ 'putLimitOrder' ],`
		`    extensions: { title: 'Not enough balance',`
		`                  code: 'NOT_ENOUGH_BALANCE' } } ],`
		`    data:       }`

7	*Name of the method/methods*	**putMarketOrder**
	Cause of error	There are not enough (not placed) limit orders placed on the market for the specified currency pair.

	Query example	`query: 'mutation{putMarketOrder(market: "BTCUSDT", side:`
		` "asks", amount: "1") {id, status, market, ctime, mtime,`
		` ftime, type,side, price, amount, taker_fee, maker_fee,`
		` deal_fee, deal_stock, deal_money, left}}',`

	Query result	`{ errors: [ { message: 'There is not enough liquidity to`
		`execute an order',`
		`    path: [ 'putMarketOrder' ],`
		`    extensions: { title: 'Not enough liquidity',`
		`                  code: 'NOT_ENOUGH_LIQUIDITY' } } ],`
		`    data:       }`

8	*Name of the method/methods*	**chart, orderHistory**
	Cause of error	Invalid time transfer format, and Unix Timestamp format was not used.

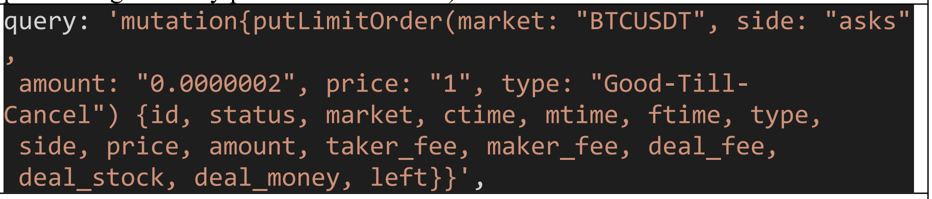
	Query example	`query: '{chart(market: "BTCUSDT", startTime: 1617235200,`
		` endTime: 1619827200, interval: 86400){time, open, high,`
		` low, close, volume} }'`

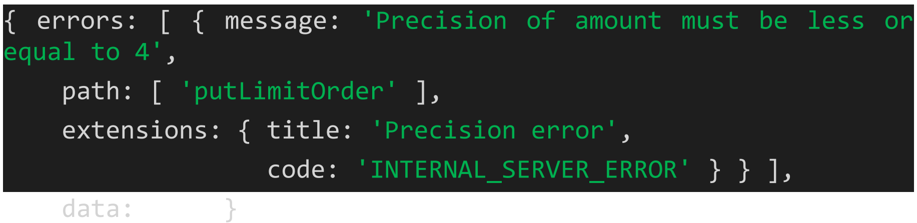
	Query result	`{ errors: [ { message: "Cannot read property 'value' of`
		`undefined",`
		`    path: [ 'chart' ],`
		`    extensions: { code: 'INTERNAL_SERVER_ERROR' } } ],`
		`    data:       }`

9	*Name of the method/methods*	**pendingOrderDetail**
	Cause of error	Error when receiving information about order (for example, order number does not exist).

	Query example	``query: `{pendingOrderDetail(market: "BTCUSDT", orderId: 12345)``
		`{id, status, market, ctime, mtime, ftime, type, side, price,`
		` amount, taker_fee, maker_fee, deal_money, deal_stock,`
		`` deal_fee, left}}`,``

	Query result	`{ errors: [ { message: 'An error occurred while fetching order`
		`details',`
		`    path: [ 'pendingOrderDetail' ],`
		`    extensions: { title: 'Failed to fetch order details',`
		`                  code: 'INTERNAL_SERVER_ERROR' } } ],`
		`    data:       }`

10	*Name of the method/methods*	**putLimitOrder, putMarketOrder, putStopLimitOrder, putStopOrder**
	Cause of error	Value accuracy (number of decimal places or the amount of minimum volume for performing currency purchase transaction) is exceeded.
	Query example	 query: 'mutation{putLimitOrder(market: "BTCUSDT", side: "asks" , amount: "0.0000002", price: "1", type: "Good-Till- Cancel") {id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_money, left}}',

	Query result	
		{ errors: [{ message: 'Precision of amount must be less or
		equal to 4',
		    path: ['putLimitOrder'],
		    extensions: { title: 'Precision error',
		                  code: 'INTERNAL_SERVER_ERROR' } }],
		    data: }

11	*Name of the method/methods*	**any method**
	Cause of error	Authorization error (keys or web-site address are incorrectly entered).

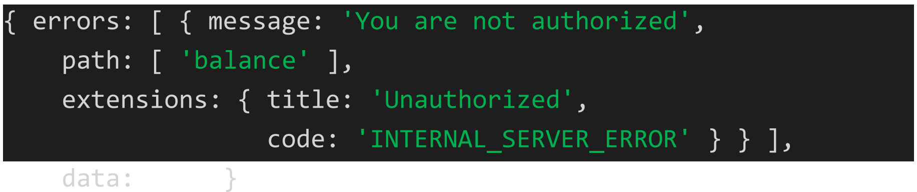
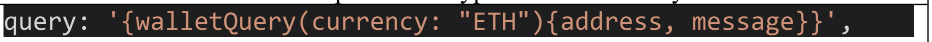
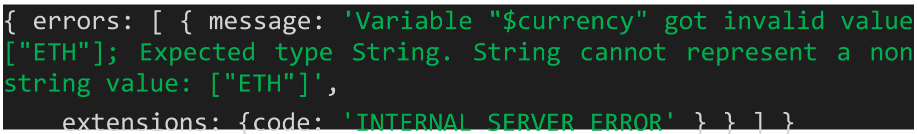
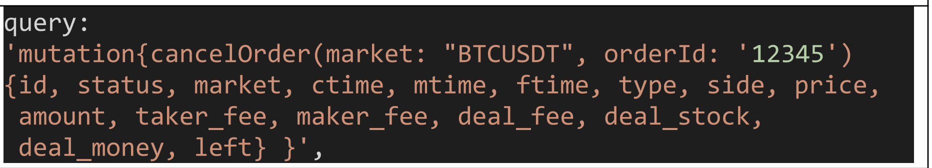
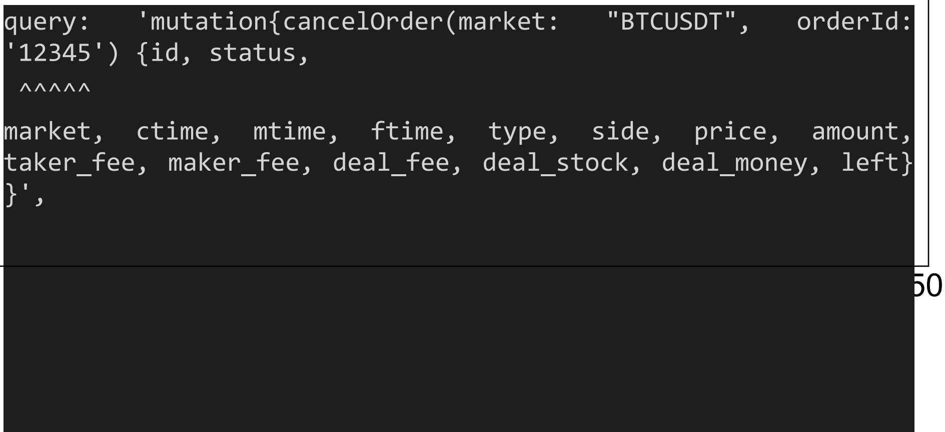
	Query result	
		{ errors: [{ message: 'You are not authorized',
		    path: ['balance'],
		    extensions: { title: 'Unauthorized',
		                  code: 'INTERNAL_SERVER_ERROR' } }],
		    data: }

Table 31 – Description of the system errors occurring in API methods execution

No.	Characteristics of errors	Examples of system errors
1	*Name of the method/methods*	**walletQuery**
	Cause of error	Refused in wallet address receipt for the crypto currencies entry.
	Query example	 query: '{walletQuery(currency: "ETH"){address, message}}',
	Query result	 { errors: [{ message: 'Variable "$currency" got invalid value ["ETH"]; Expected type String. String cannot represent a non string value: ["ETH"]', extensions: {code: 'INTERNAL_SERVER_ERROR' } }] }
2	*Name of the method/methods*	**any method**
	Cause of error	Invalid parameter record format. In example, the orderId parameter is assigned with the value type "String" instead of "Integer".
	Query example	 query: 'mutation{cancelOrder(market: "BTCUSDT", orderId: '12345') {id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_money, left} }',
	Query result	 query: 'mutation{cancelOrder(market: "BTCUSDT", orderId: '12345') {id, status, ^^^^^ market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_money, left} }',

		SyntaxError: Unexpected number
		    at new Script (vm.js:85:7)
		    at createScript (vm.js:266:10)
		    at Object.runInThisContext (vm.js:314:10)
		    at Module._compile (internal/modules/cjs/loader.js:698:28)
		    at Object.Module._extensions..js (internal/modules/cjs/loader.js:749:10)
		    at Module.load (internal/modules/cjs/loader.js:630:32)
		    at tryModuleLoad (internal/modules/cjs/loader.js:570:12)
		    at Function.Module._load (internal/modules/cjs/loader.js:562:3)
		    at Function.Module.runMain (internal/modules/cjs/loader.js:801:12)
		    at internal/main/run_main_module.js:21:11

2. Annex A. HTTP request for Stock Exchange API

```
const crypto = require('crypto')
const request = require('request')

const apiKey = 'Enter public key Token'
const apiSecret = 'Enter private key Token'
const nonce = (Date.now() / 1000) | 0 // В секундах
const body = {
    query: '{balance{currency, available, frozen}}',
    query: '{balanceHistory(currency: "BTC", transactionTypes: ["deposit", "trade_fee
"]){time, currency, change, transactionType, detailInfo {left, side, price, market, f
ee_amount, deal_amount, total_amount}}}',
    query: 'mutation{cancelOrder(market: "BTCUSDT", orderId: 12345) {id, status, mark
et, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_fee, d
eal_stock, deal_money, left} }',
    query: '{chart(market: "BTCUSDT", startTime: 1579460400, endTime: 1579496768, int
erval: 86400){time, open, high, low, close, volume} }'
    query: '{currencyList{name, prec}}',
    query: '{marketList{name, stock, money, fee_prec, stock_prec, money_prec, min_amo
unt}}',
    query: '{marketToday(currency: "BTC"){market, stock, money, open, high, low, last
Price, percentChange, volume}}'
    query: '{orderBook(market: "BTCUSDT", limit: 50, precision: "10"){asks{price, amo
unt}, bids{price, amount}}}',
    query: '{orderHistory(market: "BTCUSDT", startTime: 1579493168, endTime: 15795937
00, offset: 0, limit: 50, side: "all") {id, status, market, ctime, mtime, ftime, type
, side, price, amount, taker_fee, maker_fee, deal_money, deal_stock, deal_fee, left}}
',
    query: '{pendingOrders(market: "BTCUSDT", offset: 0, limit: 100){id, parentId, st
atus, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, d
eal_fee, deal_stock, deal_money, left, triggerType, triggerPrice}}',
    query: '{pendingOrderDetail(market: "BTCUSDT", orderId: 3909654){id, status, mark
et, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fee, deal_money,
 deal_stock, deal_fee, left}}',
    query: '{pendingSummary{market, pendingOrders}}',
    query: 'mutation{putLimitOrder(market: "BTCUSDT", side: "asks", amount: "0.0001",
 price: "1", type: "Good-Till-
Cancel") {id, status, market, ctime, mtime, ftime, type, side, price, amount, taker_f
ee, maker_fee, deal_fee, deal_stock, deal_money, left}}',
    query: 'mutation{putMarketOrder(market: "BTCUSDT", side: "asks", amount: "1") {id
, status, market, ctime, mtime, ftime, type, side, price, amount, taker_fee, maker_fe
e, deal_fee, deal_stock, deal_money, left}}',
    query: 'mutation{putStopLimitOrder(market: "BTCUSDT", side: "asks", amount: "1",
price: "10", type: "Good-Till-
Cancel", triggerType: "ask", triggerPrice: "15"){id, status, market, ctime, mtime, ft
ime, type, side, price, amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_mone
y, left, triggerType, triggerPrice}}',
    query: 'mutation{putStopOrder(market: "BTCUSDT", side: "asks", amount: "1", trigg
erType: "ask", triggerPrice: "15") {id, status, market, ctime, mtime, ftime, type, si
de, price, amount, taker_fee, maker_fee, deal_fee, deal_stock, deal_money, left, trig
gerType, triggerPrice}}'
    query: '{tradeHistory(market: "BTCUSDT", limit: 10, offset: 100){id, time, price,
 amount, type}}',
    query: '{walletCommissions {name, commission}}',
    query: '{walletQuery(currency: "BTC"){address, message}}',
}

const gurl = 'https://bizonex.com/api/v1/'

const signature = `${gurl}${nonce}${JSON.stringify(body)}`
const shex = crypto
    .createHmac('sha512', Buffer.from(apiSecret, 'hex'))
    .update(signature)
    .digest('hex')

const options = {
    contentType: 'application/json',
    url: 'https://bizonex.com/api/v1/'
    headers: {
        nonce,
        apiKey,
        signature: shex,
    },
    body,
    json: true,
}

request.post(options, (error, response, body) => {

      console.dir(response.body,{depth:10})
})
```

3. Annex B. Websocket - request for Stock Exchange API

```
const { WebSocketLink } = require('apollo-link-ws')
const crypto = require('crypto')
const gql = require('graphql-tag').default
const { SubscriptionClient } = require('subscriptions-transport-ws')
const ws = require('ws')
const { execute } = require('apollo-link')

const apikey = 'Enter public key Token'
const apiSecret = 'Enter private key Token'

const nonce = (Date.now() / 1000) | 0 // В секундах

const requestTable = {
        marketUpdated: {
                balanceUpdated: {
                query: `
                subscription balanceUpdated($currencyList: [String!]!) {
                        balanceUpdated(currencyList: $currencyList) {
                                currency
                                available
                                frozen
                        }
                }`,
                variables: { currencyList: ['BTC'] },
        },
        chartUpdated: {
                query: `
                subscription chartUpdated($market: String!, $interval: Int!) {
                        chartUpdated(market: $market, interval: $interval) {
                                time
                                open
                                high
                                low
                                close
                                volume
                        }
                }`,
                variables: { market: 'BTCUSDT', interval: 60 },
        },
        dealsUpdated: {
                query: `
                subscription dealsUpdated($market: String!) {
                        dealsUpdated(market: $market) {
                                id
                                time
                                price
                                amount
                                type
                        }
                }`,
                variables: { market: 'BTCUSDT' },
        },
        query: `
        subscription marketUpdated($currency: String!) {
                        marketUpdated(currency: $currency) {
                                market
                                stock
                                money
                open
                high
                low
                lastPrice
                percentChange
                volume
            }
        }`,
        variables: { currency: 'BTC' },
    },
    orderBookUpdated: {
        query: `
        subscription orderBookUpdated($market: String!, $limit: Int! = 50,
$precision: String! = "0") {
            orderBookUpdated(market: $market, limit: $limit, precision:
$precision) {
                asks {
                    price
                    amount
                }
                bids {
                    price
                    amount
                }
                entirety
            }
        }`,
        variables: { market: 'BTCUSDT', limit: 50, precision: '0.0001' },
    },
    orderUpdated: {
        query: `
        subscription orderUpdated($markets: [String!]) {
            orderUpdated(markets: $markets) {
                id
                status
                market
                ctime
                mtime
                ftime
                type
                side
                price
                amount
                taker_fee
                maker_fee
                deal_fee
                deal_stock
                deal_money
                left
                triggerType
                triggerPrice
            }
        }`,
        variables: { markets: ['BTCUSDT'] },
    },
    stateUpdated: {
        query: `
        subscription stateUpdated($market: String!) {
            stateUpdated(market: $market) {
                open
                market
                stock
                money
                high
                low
                lastPrice
                percentChange
                volume
            }
        }`,
        variables: { market: 'BTCUSD' },
    },
}

const wsUrl = 'wss://bizonex.com/api/v1/'
const httpUrl = 'https://bizonex.com/api/v1/'

const getServerSignature = ({ nonce, body, apiSecret }) => {
    const data = `${httpUrl}${nonce}${JSON.stringify(body)}`

    const serverSignature = crypto
        .createHmac('sha512', Buffer.from(apiSecret, 'hex'))
        .update(data)
        .digest('hex')

    return serverSignature
}

const request = requestTable.stateUpdated

const signature = getServerSignature({
    nonce,
    body: request.query,
    apiSecret,
})

const options = {
    headers: { nonce, apikey, signature },
    body: request.query,
}

const createSubscriptionObservable = (wsUrl, query, variables) => {
    const link = new WebSocketLink(
        new SubscriptionClient(
            wsUrl,
            {
                connectionParams: options,
                connectionCallback: error => {
                    console.log(`Subscribed  successfully,  waiting  for
messages...`, error)
                },
            },
            ws
        )
    )

    return execute(link, {
        query,
        variables,
    })
}
```

```
const subscribe = () => {
	const subscriptionObservable = createSubscriptionObservable(
		wsUrl,
		gql`
			${request.query}
		`,
		request.variables
	)

	subscriptionObservable.subscribe(eventData => {
		console.log(`Subscribed successfully, waiting for messages...`)
		console.dir(eventData, { depth: 10 })
	})
}

subscribe()
```

4. LIST OF ABBREVIATIONS AND SYMBOLS

API (Application Programming Interface)	a set of public properties and methods for interaction with other objects in the application.
HTTP (HyperText Transfer Protocol)	data transfer Protocol initially intended for transmitting hypertext documents (documents that may contain the links allowing to navigate to other documents).

5. LIST OF TERMS AND DEFINITIONS

API-keys (application programming interface keys) is a set of public and secret Token keys, where the new block chains are created based on confirmation of transactions with a secret key.

ICO (Initial coin offering) is a form for investments raising as a fixed number of new units of cryptocurrency selling to investors received due to a single or accelerated issue.

Token – bot creation having access to a pair of public and private keys. Bot provides specified types of transactions execution on the Stock Exchange on behalf of the user.

Websocket is a communications Protocol over a TCP connection intended for real-time interaction between a web browser and a web server. Unlike the HTTP Protocol, data is sent via the Websocket Protocol when any event to which subscription was made occurs. Until connection closure, subscription remains active and is waiting for server responses.

Block chain is a decentralized database where data storage devices are not connected to a shared server. Users can only edit that parts of block chain they have access to, based on the keys provided. Encryption ensures copies of distributed block chain synchronization for all users.

Cryptocurrency is an internal unit of currency obtained using mining and linked to one or more wallets. When performing any operations with cryptocurrency, operations are recorded in the blockchain.

Limit order is a type of order that indicates the maximum price to broker at which the user is ready to buy or the minimum price at which the user is ready to sell any currency.

Market order is a type of order executed immediately when a limit order with the best current price is found.

Mining is a series of calculations with parameters iteration to find "hash" with the specified properties required to ensure cryptocurrency platforms functioning.

Maker is a part of traders who operate on trading platforms and provide additional liquidity for the asset, contributing to quotations shift towards increase or decrease. Traders place pending orders the price level of which is higher or lower than the current values. The Stock Exchange encourages this category of traders with no transaction fees or by reducing interest to a minimum.

Taker is a part of traders operating on trading platforms and reducing asset liquidity due to decrease in a number of offers placed. Traders perform trading transactions immediately upon pending order placement with the relevant conditions at a market price.

Forging / **Minting** is a new block creation in cryptocurrencies based on the ownership share confirmation with ability to receive remuneration as new units and commission fees.

Hash table is a data structure implementing associative array interface and allowing to store pairs (key, value) and perform three types of transactions: new pair addition, search and pair deletion by key. Each hash is unique, not related by value to the previous one.

Issue is an issue of new money in the circulation.

6. Versions of the documents

No.	Version number and creation date	Version characteristics
1	Version 1.0 dd. 07.03.2019	The first section describes the process of Token creation, setup and deletion on the Stock Exchange page in "User information" section. The second section shows the process of Postman and Visual Studio Code applications running and setting up to make queries using API-keys section. The third section describes HTTP queries methods allowing to perform transactions with API-keys. Annex A contains connection settings and a list of methods. The document contains the lists of abbreviations and symbols, terms and definitions.
2	Version 1.1 dd. 04.12.2019	The first section describes the process of Token creation, setup and deletion on the Stock Exchange page in "User information" section. The third section describes HTTP and Websocket queries methods allowing to perform transactions with API-keys. Annex A contains a list of connection settings for the HTTP Protocol. Annex contains a list of connection settings for the Websocket Protocol. The document contains the lists of abbreviations and symbols, terms and definitions. Changes made: 1. Section on SW application installation and setup is deleted; 2. Websocket methods are described. 3. Examples and drawings provided have been changed according to the program interface update. 4. Fixed hyperlinks to resource addresses.
3	Revision 1.2 dd. 12.02.2020	The first section describes the process of Token creation, setup and deletion on the Stock Exchange page in "User information" section. The third section describes HTTP and Websocket queries methods allowing to perform transactions with API-keys. Annex A contains a list of connection settings for the HTTP Protocol. Annex contains a list of connection settings for the Websocket Protocol. The document contains the lists of abbreviations and symbols, terms and definitions. Changes made: 1. The structure of section 2 has been changed due to the new HTTP methods addition and requests parameters update; 2. Section describing business-logic errors and system errors occurring when executing methods has been added. 3. Section describing documents versions has been added; 4. The list of currencies in query results has been updated; 5. Descriptions of some parameters have been corrected due to changes in accuracy (stock_prec, money_prec) and minimum amount of currency purchase transactions (min_amount).