



ИСПОЛЬЗОВАНИЕ API НА БИРЖЕ EXS.CASH

Версия 1.1

ОПИСАНИЕ СИСТЕМЫ

2019

Аннотация

Данный документ описывает структуру ввода, обработки, хранения и удаления информации по API-keys на Бирже EXS.CASH.

Документ включает три раздела. В первом приведен процесс создания, настройки и удаления Token на странице Биржи «Личные данные». Во втором показан процесс запуска и настройки приложений Postman, Visual Studio Code для выполнения запросов по разделу API-keys. В третьем подробно рассмотрены методы запроса, позволяющие выполнять операции с API-keys.

Приложение А содержит текст запроса, написанный на языке Java Script.

К документу приложены списки сокращений и обозначений, терминов и определений.

Документ выпущен в версии 1.1.

СОДЕРЖАНИЕ

Общие сведения.....	4
1 Настройка программ - ботов на Бирже EXS.CASH.....	5
1.1 Личные данные пользователя	5
1.2 Прохождение процедуры верификации	6
1.3 Функциональное предназначение Api-keys (Token)	11
1.4 Создание Token	11
1.5 Раскрытие Token.....	13
1.6 Удаление Token	14
2 Установка программ для тестирования запросов.....	16
2.1 Приложение Postman	16
2.1.1 Установка Postman	16
2.1.2 Настройка Postman	17
2.1.3 Выполнение запроса	19
2.2 Приложение Visual Studio Code	22
2.2.1 Установка Node.js	22
2.2.2 Установка Visual Studio Code.....	23
2.2.3 Настройка Visual Studio Code	23
2.2.4 Выполнение запроса	26
3 Описание запроса	28
3.1 Структура запроса	29
3.2 Описание методов	36
3.3 Выходные данные.....	52
Приложение А. Тест HTTP-запроса для API Биржи	56
Приложение Б. Тест Websocket -запроса для API Биржи.....	58
СПИСОК СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	62
СПИСОК ТЕРМИНОВ И ОПРЕДЕЛЕНИЙ	63

Общие сведения

EXS.CASH (далее - Биржа) работает как электронное приложение, вызываемое из окна браузера при переходе по адресу: <https://exs.cash/api/v1>.

Биржа EXS.CASH – это биржа криптовалют, позволяющая производить торговые операции между фиатными средствами (доллары, евро, рубли) и видами криптовалют на основании создания ордеров по покупке или продаже.

Основные виды обмена между валютными средствами:

- 1) фиатные средства на криптовалюту;
- 2) криптовалюта на фиатные средства;
- 3) пары криптовалют между собой.

ПРИМЕЧАНИЕ: на Бирже взимается комиссия в размере 0,2% от каждой проведенной сделки по ордерам продажи или покупки.

В документации рассмотрены процессы создания, настройки и удаления Token на странице Биржи «Личные данные», описаны запуск и настройка приложений Postman, Visual Studio Code для выполнения запросов по разделу «API КЛЮЧИ» и приведены методы запроса, позволяющие выполнять операции с API-keys.

Формат выполнения запросов - «POST». Запросы используют публичные или приватные методы. Для осуществления запроса к приватным методам, требуется наличие действующего публичного ключа и, соответствующего ему, секретного ключа.

1 Настройка программ - ботов на Бирже EXS.CASH

Программы-боты (далее - боты) предназначены для выполнения сделок на криптовалютной бирже от имени пользователя.

Биржевые боты для автоматической торговли криптовалютой функционируют по особым алгоритмам, которые создаются на основании анализа числа потенциальных убытков и прибылей, проведенных за расчетный период. На основе проведенного анализа формулируются правила торговой стратегии робота для Биржи криптовалют.

Биржевые боты, действующие в рамках заданного алгоритма, имеют ряд преимуществ перед трейдером при осуществлении сделок:


- скорость принятия решений при проведении ордеров;
- возможность анализа неограниченного количества пар;
- круглосуточное выполнение операций.

При регистрации на Бирже EXS.CASH создание бота доступно на странице «Личные данные» в разделе «API КЛЮЧИ». В разделе может быть создан новый API-key (Token). Количество доступных Token операций ограничено набором настроек, заданных пользователем.

Созданный биржевой бот работает только на одной торговой площадке - EXS.CASH, использование ботов бесплатно для пользователя.

Комиссия в размере 0,2% взимается только за проведенные сделок по продаже и покупке.

1.1 Личные данные пользователя

Переход к странице «Личные данные» происходит при нажатии иконки  в верхнем правом углу экрана Биржи EXS.CASH (при переходе к разделу цвет иконки меняется с белого на золотой).

Список разделов «Личные данные»:

1 «Пользовательская информация» - общая информация о статусе пользователя:

- имя пользователя (login);
- адрес электронной почты;
- номер телефона;
- уровень доступа согласно верификации (подробнее см. раздел «FAQ > Настройки аккаунта > Этапы верификации, функциональные возможности»);
- доступный объем для вывода криптовалют в BTC;
- доступный объем для вывода фиатных денег в USD;
- дата обновления данных.

2 «Верификация» - личные данные, предоставляемые пользователем для прохождения процедуры верификации:

- Персональная информация.
- Адрес.
- Информация о доходах.
- Документы.
- Фото и соглашения.

В случае предоставления недостоверной или неполной информации верификация не будет пройдена.

Информацию в разделе можно изменить до момента проверки данных, после прохождения процедуры раздел «Верификация» закрыт для редактирования и просмотра. При изменении данных необходимо обратиться в службу поддержки пользователей.

3 «Безопасность» - настройки двухфакторной аутентификации и изменения пароля пользователя.

4 «API ключи» - форма создания "ботов", выполняющих операции от имени пользователя.

Переход по закладкам производится при переходе к меню настроек страницы «Личные данные». Активная закладка выделяется золотым цветом (Рисунок 1).

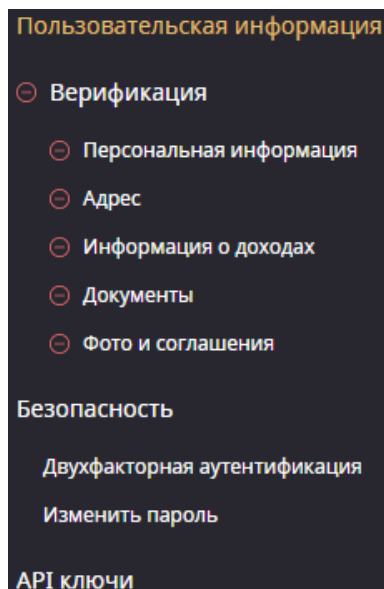


Рисунок 1 – Меню настроек страницы «Личные данные»

В документе подробно рассмотрены только разделы «Верификация» и «API КЛЮЧИ». Описание полного функционала Биржи EXS.CASH выходит за рамки данного документа.

1.2 Прохождение процедуры верификации

Верификация - процедура проверки подлинности предоставленных личных данных (персональная информация, адрес, информация о доходах) и скан-копий документов (документы, фото и соглашения пользователя).

После регистрации пользователю присваивается первый уровень верификации (поле «Уровень доступа»), последующие уровни (2 - 6) доступны после заполнения закладок раздела «Верификация» и подтверждения достоверности предоставленных данных (подробнее см. раздел «FAQ > Настройки аккаунта > Этапы верификации, функциональные возможности»). Если пользователь не верифицирован по результатам проверки, то на его электронный почтовый ящик будет направлено письмо с дополнительной информацией.

Не верифицированный пользователь (Рисунок 3) имеет ряд ограничений при выполнении финансовых операций на Бирже EXS.CASH, до момента успешного прохождения верификации закрыт доступ к разделу «Маркет» на странице «Торговля» (в разделе приведена информация о необходимости пройти верификацию и ссылка на страницу «Личные данные», Рисунок 2).

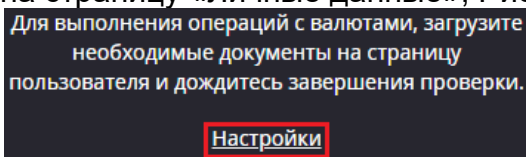
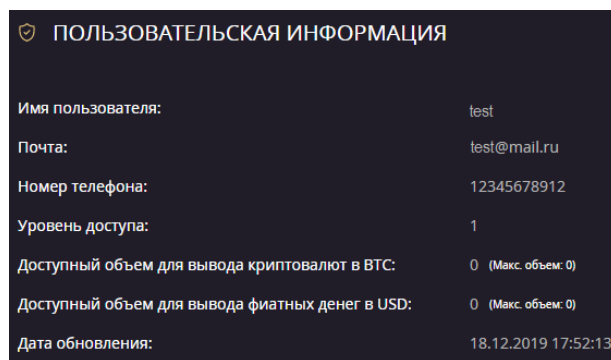


Рисунок 2 – Раздел «Маркет» после прохождения процедуры регистрации



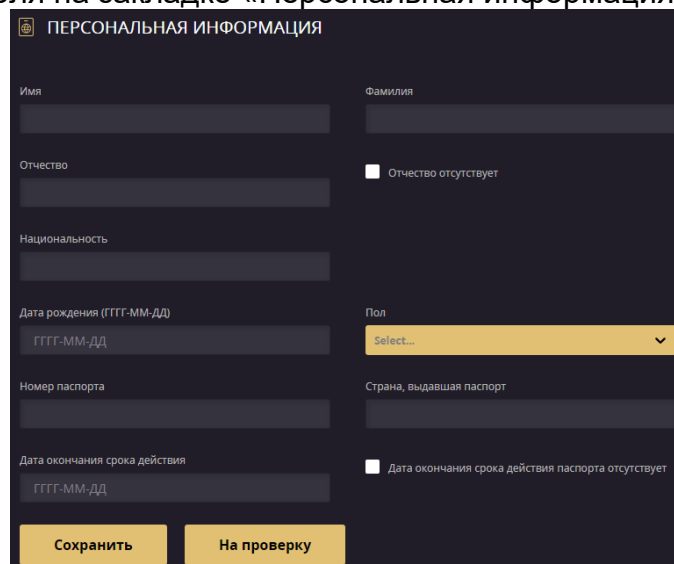
ПОЛЬЗОВАТЕЛЬСКАЯ ИНФОРМАЦИЯ	
Имя пользователя:	test
Почта:	test@mail.ru
Номер телефона:	12345678912
Уровень доступа:	1
Доступный объем для вывода криптовалют в BTC:	0 (Макс. объем: 0)
Доступный объем для вывода фиатных денег в USD:	0 (Макс. объем: 0)
Дата обновления:	18.12.2019 17:52:13

Рисунок 3 – Раздел «Пользовательская информация» после прохождения процедуры регистрации

Процедура прохождения верификации включает следующие этапы:

- *Редактирование личных данных пользователя:*

1) Ввести значения в полях ФИО, паспортные данные, пол пользователя на закладке «Персональная информация» (Рисунок 4):



ПЕРСОНАЛЬНАЯ ИНФОРМАЦИЯ

Имя: Фамилия:

Отчество: ☐ Отчество отсутствует

Национальность:

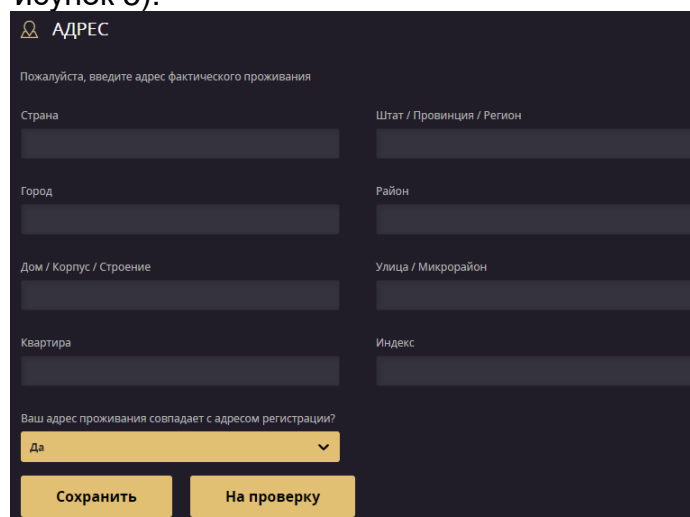
Дата рождения (ГГГГ-ММ-ДД): Пол:

Номер паспорта: Страна, выдавшая паспорт:

Дата окончания срока действия: ☐ Дата окончания срока действия паспорта отсутствует

Рисунок 4 - Закладка «Персональные данные» в формате редактирования

Указать адреса фактического проживания и регистрации на закладке «Адрес» (Рисунок 5):



АДРЕС

Пожалуйста, введите адрес фактического проживания

Страна: Штат / Провинция / Регион:

Город: Район:

Дом / Корпус / Строение: Улица / Микрорайон:

Квартира: Индекс:

Ваш адрес проживания совпадает с адресом регистрации?

Рисунок 5 - Закладка «Адрес» в формате редактирования

Выбрать один из вариантов по статусу занятости и указать контактные данные на закладке «Информация о доходах» (Рисунок 6):

ИНФОРМАЦИЯ О ДОХОДАХ

Статус занятости
Работник

Индустрия

Должность / Описание

Название компании / организации

Страна компании / организации

Штат / область / регион

Город / населенный пункт

Улица / Микрорайон

Почтовый индекс

Сохранить На проверку

ИНФОРМАЦИЯ О ДОХОДАХ

Статус занятости
Частный предприниматель

Индустрия

Должность / Описание

Название компании / организации

Страна компании / организации

Штат / область / регион

Город компании

Город / населенный пункт

Почтовый индекс

Сохранить На проверку

ИНФОРМАЦИЯ О ДОХОДАХ

Статус занятости
Другое

Введите ваш статус

Ваш источник дохода

Сохранить На проверку

Рисунок 6 - Закладка «Информация о доходах» в формате редактирования

2) Для передачи данных на сервер нажать кнопку «Сохранить», в нижней части экрана отобразится сообщение об успешной загрузке информации (Рисунок 7).

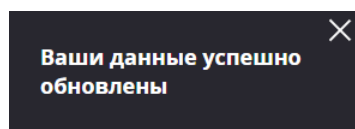


Рисунок 7 – Информационное сообщение при сохранении информации

После отправления информации на проверку специалисту (кнопка «На проверку») на почтовый ящик пользователя будет направлено электронное письмо о проверке подлинности данных (Рисунок 8).

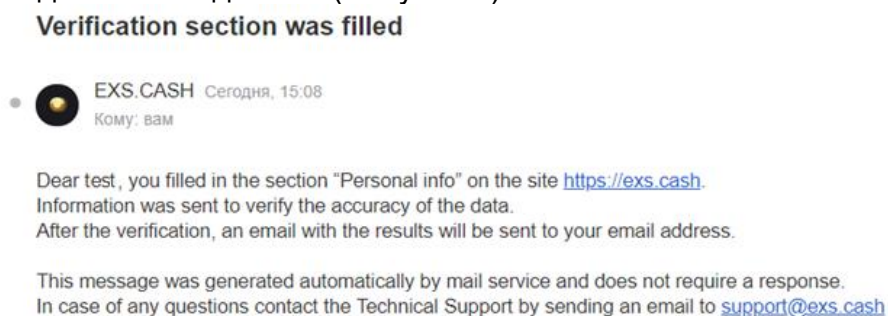


Рисунок 8 – Электронное сообщение об отправлении одного из разделов «Верификация» на проверку подлинности данных

- *Добавление скан-копий документов:*

3) На закладке «Документы» напротив одного/ нескольких документов нажать кнопку «Выберите файл» для указания пути до скан-копии документа и выполнить загрузку при помощи кнопки «На проверку». Максимальный размер загружаемого файла – 5МБ.

При сохранении данных процесс загрузки отобразится в виде одного из статусов (Рисунок 9):

- Ваш файл загружается, пожалуйста, подождите;
- Готово к загрузке на сервер;
- Успешно загружено на сервер.

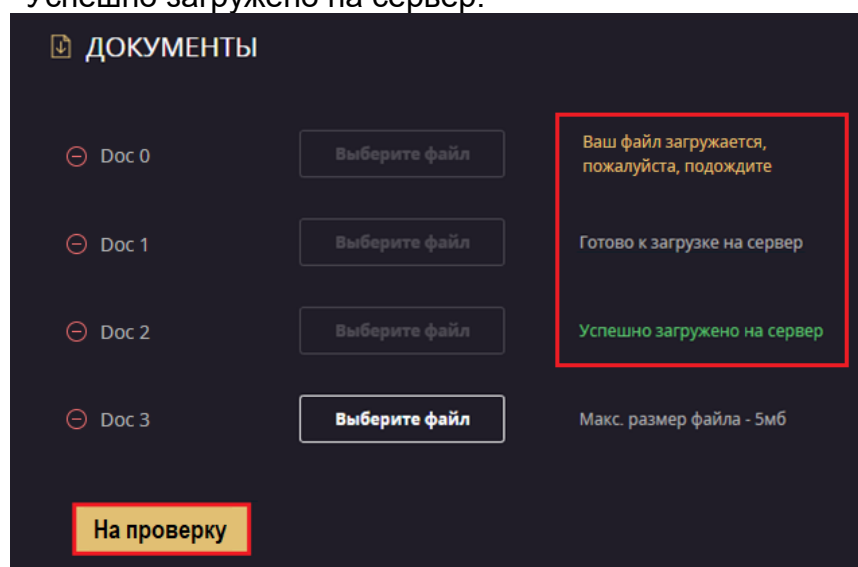


Рисунок 9 - Закладка «Документы» при сохранении информации

ПРИМЕЧАНИЕ: Список документов, предоставляемых пользователем (минимальное количество – 2 документа):

- паспорт гражданина государства/ заграничный паспорт гражданина (обязательно предоставляется один из документов);
- военный билет/ водительское удостоверение (обязательно предоставляется один из документов).

4) На закладке «Фото и соглашения» необходимо добавить фотографию пользователя в указанном ниже формате и проставить галочки о предоставлении достоверных данных.

- Фотография включает следующие элементы (Рисунок 10):
- фотография пользователя;
 - паспорт или другой документ, удостоверяющий личность (один из файлов с закладки «Документы»);
 - запись EXS.CASH;
 - текущая дата;
 - подпись пользователя.
- Максимальный размер загружаемого файла – 5МБ.

ФОТО И СОГЛАШЕНИЯ

Пожалуйста, предоставьте фотографию с вашим паспортом или другим документом являющимся удостоверением личности.

На этом селфи также должен быть лист с записью «EXS.CASH», сегодняшняя дата, и ваша подпись.

Убедитесь, что ваше лицо будет хорошо видно и все паспортные данные четко читаемы.

Выберите файл Selfie Макс. размер файла - 5мб

☐ Я подтверждаю, что предоставленная мной информация является достоверной и полной

☐ Я согласен и подтверждаю, что при необходимости предоставляю все необходимые подтверждения данных

На проверку

Рисунок 10 - Закладка «Документы» при сохранении информации

- Проверка и подтверждение информации, загруженной пользователем (выполняется специалистами Биржи EXS.CASH).

5) При условии подтверждения достоверности данных пользователь получает новый уровень доступа (подробнее см. раздел «FAQ > Настройки аккаунта > Этапы верификации, функциональные возможности») (Рисунок 11). В случае несовпадения предоставленных данных или возникновения подозрений в подлинности предоставленных документов информация может быть дополнительно запрошена по электронному адресу пользователя (адрес регистрации учетной записи).

ПОЛЬЗОВАТЕЛЬСКАЯ ИНФОРМАЦИЯ

Имя пользователя:	test
Почта:	test@mail.ru
Номер телефона:	12345678912
Уровень доступа:	6
Доступный объем для вывода фиатных денег в USD:	0 (Макс. объем: 0)
Дата обновления:	18.12.2019 17:52:13

Рисунок 11 - Раздел «Пользовательская информация» после прохождения процедуры верификации

При изменении документов пользователя необходимо обратиться в «Службу технической поддержки».

Сроки проведения верификации при первичном/ повторном прохождении процедуры определяются количеством заявок и качеством предоставленной документации.

В случае возникновения вопросов обращаться в «Службу технической поддержки» по адресу: support@exs.cash.

ПРИМЕЧАНИЕ. До момента прохождения верификации информация о пользователе может быть изменена, после проведения раздел «Верификация» недоступен для просмотра и редактирования.

1.3 Функциональное предназначение Api-keys (Token)

API-keys (Token) - ключи доступа к информации пользователя, предоставленные программе - боту для выполнения ряда операций от имени пользователя по чтению или записи данных на Бирже EXS.CASH.

По созданному API-key производится 1 запрос в секунду, запрос может включать несколько операций, указанных в Token пользователя.

При использовании Token проводится проверка законности применения (доступность логину используемого API-key) и подлинности ключа (проверка совпадения пары публичного и секретного ключей). При установлении недостоверности применение Token может быть заблокировано. Разблокирование происходит в случае обращения пользователя – владельца ключа в «Службу технической поддержки».

ПРИМЕЧАНИЕ: пользователь несет полную ответственность за передачу публичного и секретного ключей сторонним лицам и финансовые риски при проведении любых операций сторонними лицами с применением его API-key. В случае установления факта нарушения использования Token выставленные ордера, ожидающие исполнения, могут быть заблокированы, средства по операциям заморожены до момента установления законности проведенных сделок.

1.4 Создание Token

Для создания нового Api-ключа пользователю необходимо выполнить следующие действия:

- В разделе «API КЛЮЧИ» установить уровень доступа программе-боту по новому ключу:

1) Ввести имя нового Token (login).

2) Проставить галочки в выбранных полях.

- Информация аккаунта (тип - чтение) - доступ на чтение к разделам Биржи.

- История аккаунта (тип - чтение) - доступ на чтение к истории финансовых операций, проводимых пользователем.

- Управление ордерами (тип - чтение) - чтение информации по выставленным ордерам по продаже или покупке.

- Управление ордерами (тип - запись) - выполнение встречных ордеров по продаже или покупке от имени пользователя.

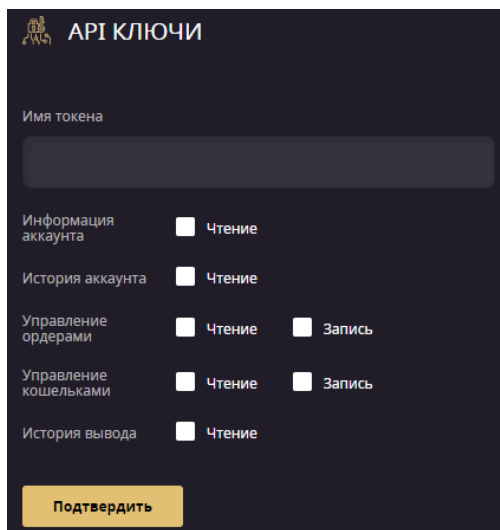
- Управление кошельками (тип - чтение) - вывод информации по количеству наличных средств по видам валют пользователя.

- Управление кошельками (тип - запись) - перевод личных денежных средств пользователя между различными видами валют, необходимый для проведения операций на Бирже от имени пользователя.

- История вывода (тип - чтение) - доступ на чтение к информации о введенных и выведенных наличных средствах пользователя.

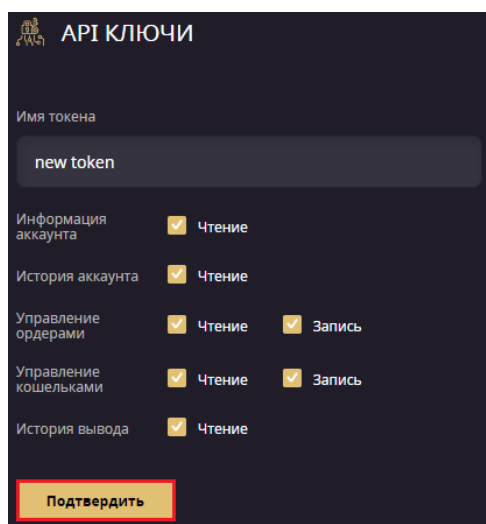
3) Нажать кнопку «Подтвердить».

Внешний вид формы создания нового Арі-ключа (Рисунок 12, Рисунок 13):



The screenshot shows a dark-themed form titled 'АРИ КЛЮЧИ'. At the top is a text input field labeled 'Имя токена'. Below it is a list of permissions, each with an unchecked checkbox and the label 'Чтение': 'Информация аккаунта', 'История аккаунта', 'Управление ордерами', 'Управление кошельками', and 'История вывода'. The 'Управление ордерами' and 'Управление кошельками' items also have an unchecked checkbox labeled 'Запись'. At the bottom is a yellow button labeled 'Подтвердить'.

Рисунок 12 - Форма создания нового Арі-ключа (Token)



This screenshot shows the same 'АРИ КЛЮЧИ' form, but with the 'Имя токена' field filled with 'new token'. All the permission checkboxes are now checked with yellow checkmarks. The 'Подтвердить' button at the bottom is highlighted with a red rectangular border.

Рисунок 13 - Форма создания нового Арі-ключа (Token) с заполненными полями

ПРИМЕЧАНИЕ. Настройки созданного Token не могут быть отредактированы.

- Изучить информационное сообщение из направленного письма для подтверждения создания нового Token.

Сообщение о направлении электронного письма по адресу регистрации пользователя отобразится на экране после нажатия кнопки «Подтвердить» (Рисунок 14).

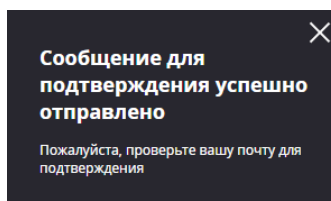


Рисунок 14 - Информационное сообщение

- Открыть электронное письмо, которое содержит ссылку на подтверждение выполнения действия и информацию о времени действия ссылки.

Ссылку необходимо скопировать в окно браузера или перейти по ней. После истечения установленного времени (1 час) ссылка будет не доступна, необходимо повторить действия для создания нового Арі-ключа (Рисунок 15).

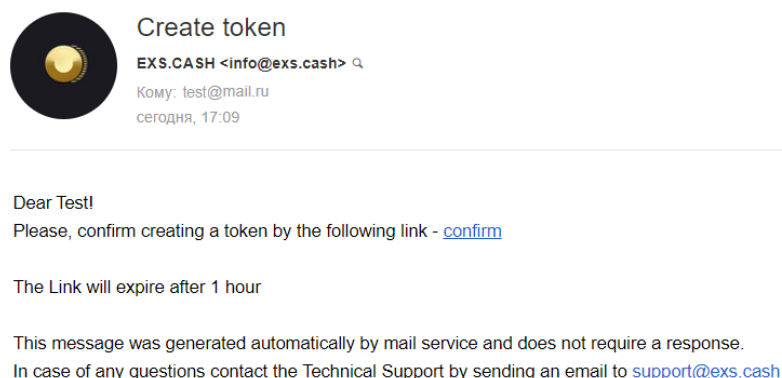


Рисунок 15 - Электронное сообщение со ссылкой на создание Арі-ключа

После перехода по ссылке в разделе «АРІ КЛЮЧИ» будет создан новый Token, секретный ключ показан до момента обновления страницы (Рисунок 16).

#1	Имя токена:	new token	Права токена:	
	Публичный ключ:	Код	Информация аккаунта	<input checked="" type="checkbox"/> Чтение
	Секретный ключ:	Код	История аккаунта	<input checked="" type="checkbox"/> Чтение
	Время создания:	25 июня 2019 г., 18:05:33	Управление ордерами	<input checked="" type="checkbox"/> Чтение <input checked="" type="checkbox"/> Запись
	Удалить токен:	Удалить	Управление кошельками	<input checked="" type="checkbox"/> Чтение <input checked="" type="checkbox"/> Запись
			История вывода	<input checked="" type="checkbox"/> Чтение

Рисунок 16 - Новый Token

1.5 Раскрытие Token

Для отображения секретного Арі-ключа пользователю необходимо выполнить следующие действия:

- Нажать кнопку «Показать секретный код» в поле «Секретный ключ» раздела «АРІ КЛЮЧИ». На экране отобразится информационное сообщение о направлении электронного письма по адресу регистрации пользователя для подтверждения выполнения операции (Рисунок 17).

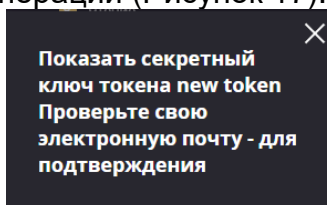


Рисунок 17 - Информационное сообщение о направлении письма

- Открыть электронное письмо, которое содержит ссылку на подтверждение выполнения действия и информацию о времени действия ссылки.

Ссылку необходимо скопировать в окно браузера или перейти по ней. После истечения установленного времени (1 часа) ссылка будет не доступна, необходимо повторить действия для открытия секретного ключа Token (Рисунок 18).

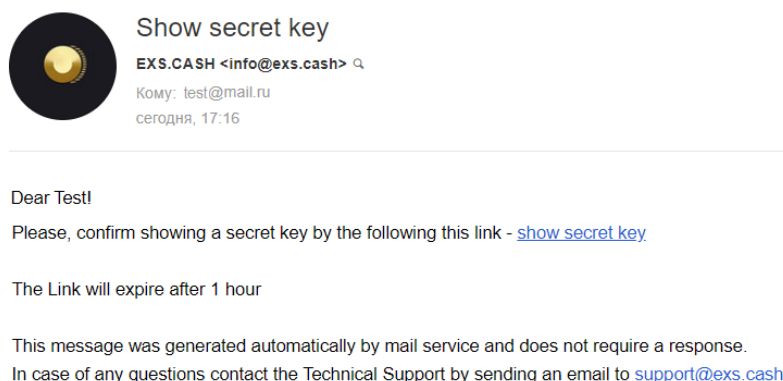


Рисунок 18 - Электронное сообщение со ссылкой на открытие секретного ключа Token

После перехода по ссылке в разделе «API КЛЮЧИ» будет показан секретный ключ Token, доступный для чтения до момента обновления страницы.

Внешний вид формы «API КЛЮЧИ» созданного Token содержит следующие поля (Рисунок 19):

- 1 Имя токена – login.
- 2 Публичный ключ – код публичного ключа.
- 3 Секретный ключ – код секретного ключа.
- 4 Время создания – информация о дате и времени создания.
- 5 Права токена – список доступных прав Token.
- 6 Удалить токен – функция удаления токена.
- 7 Права токена - уровень доступа программе-боту по Token.

#1	Имя токена:	new token	Права токена:	
	Публичный ключ:	Код	Информация аккаунта	<input checked="" type="checkbox"/> Чтение
	Секретный ключ:	Код	История аккаунта	<input checked="" type="checkbox"/> Чтение
	Время создания:	25 июня 2019 г., 18:05:33	Управление ордерами	<input checked="" type="checkbox"/> Чтение <input checked="" type="checkbox"/> Запись
	Удалить токен:	Удалить	Управление кошельками	<input checked="" type="checkbox"/> Чтение <input checked="" type="checkbox"/> Запись
			История вывода	<input checked="" type="checkbox"/> Чтение

Рисунок 19 - Внешний вид формы «API КЛЮЧИ» созданного Token

ПРИМЕЧАНИЕ: Значения полей «Публичный ключ» и «Секретный ключ» не должны предоставляться сторонним лицам.

1.6 Удаление Token

Для удаления Api - ключа пользователю необходимо выполнить следующие действия:

- Нажать кнопку «Удалить» в поле «Удалить токен».

На экране отобразится информационное сообщение о направлении электронного письма по адресу регистрации пользователя для подтверждения удаления Token (Рисунок 20).

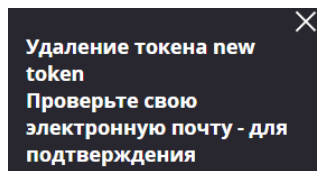


Рисунок 20 - Информационное сообщение о направлении письма

- *Открыть электронное письмо, которое содержит ссылку на подтверждение выполнения действия и информацию о времени действия ссылки.*

Ссылку необходимо скопировать в окно браузера или перейти по ней. После истечения установленного времени (1 часа) ссылка будет не доступна, необходимо повторить действия для удаления Token (Рисунок 21).

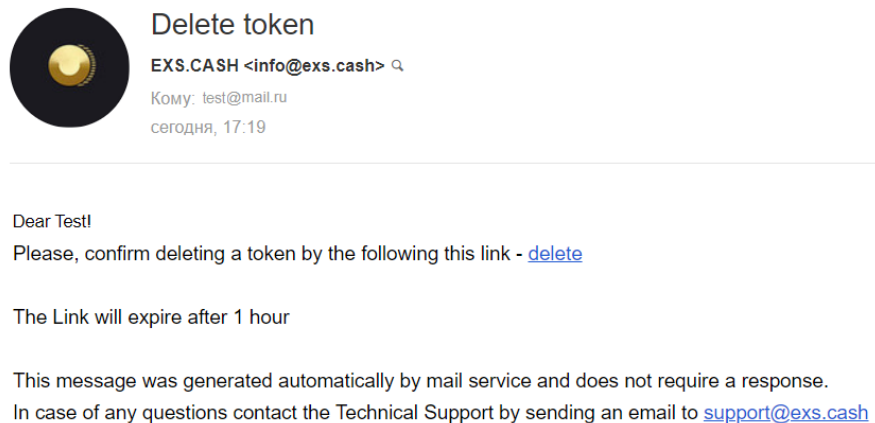


Рисунок 21 - Электронное сообщение со ссылкой на удаление Арі-ключа

2 Установка программ для тестирования запросов

Тестирование запросов может быть проведено в программе Postman (выполнения отдельных query) или в программе Visual Studio Code при использовании интерпретатора Node.js (выполнение запроса с введенными константами и дополнительными настройками).

Перед проведением тестирования следует перейти на страницу «Личные данные», загрузить документы и пройти процедуру верификации, а затем создать новый Token, которому установлены полные права на чтение и запись данных (подробнее см. раздел 1).

Передача ключей Token в программе Postman происходит на закладке «Header» (необходимо создать новые переменные), в программе Visual Studio Code - в тексте запроса (необходимо ввести Api-key, указав значения приватного и секретного ключей Token).

Для функционирования Api на Бирже реализованы публичные и приватные методы. Публичные методы могут быть выполнены без обращения к Token пользователя. Приватные методы проводят «хеширования» информации, потому дополнительно потребуется ввод действующего приватного и, соответствующего ему, секретного ключей Token и проверка синхронизации времени на компьютере пользователя со временем на сервере (на компьютере пользователя в настройках «Даты и времени» по умолчанию должна быть выбрана синхронизация времени по интернету).

Описание запроса приведено в следующем разделе на примере его выполнения в программе Visual Studio Code (подробнее см. раздел 3), полный текст запроса добавлен в приложении А (подробнее см. «Приложение А. Тест HTTP-запроса для API Биржи»).

2.1 Приложение Postman

Postman – это HTTP - клиент для тестирования и создания коллекций запросов, позволяющий улучшать производительность разработки программ. Postman представлен в вариантах прикладной программы и приложения каталога Chrome Web Store.

2.1.1 Установка Postman

Установка программы включает следующие этапы:

- Перейти на страницу сайта приложения - <https://www.getpostman.com/>, в нижней части страницы в разделе «Ресурсы» (Resources) выбрать пункт «Установка» (Downloads) (Рисунок 22).

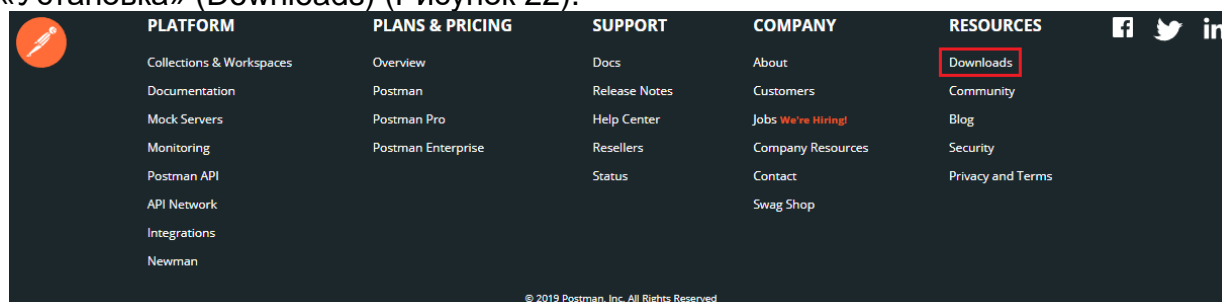


Рисунок 22 – Ссылки на ресурсы Postman на официальной странице программы

- Postman доступен как прикладная программа для Windows (32- и 64 бит), MacOS и Linux (32- и 64 бит) (Рисунок 23). После выбора одного из вариантов платформы загрузка начнется автоматически. Версия приложения для Google

Chrome может быть установлена через настройки расширения браузера. В примере показана установка для Windows 7 и более новых версий.

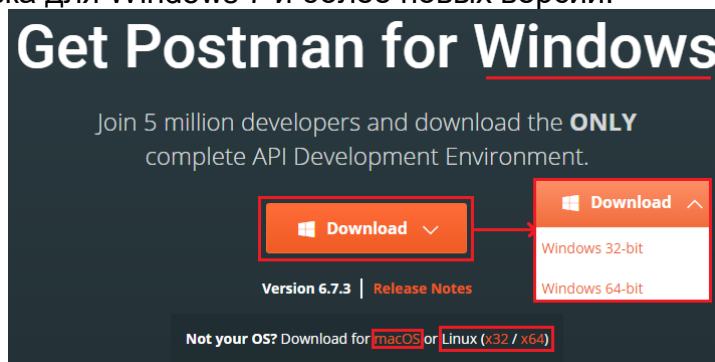


Рисунок 23 – Ссылки на установку программы на разных платформах

- Установка начинается после двойного нажатия на ярлык программы (Рисунок 24).

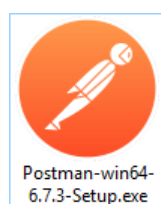


Рисунок 24 – Ярлык для установки программы

Внешний вид процесса загрузки программы (Рисунок 25):

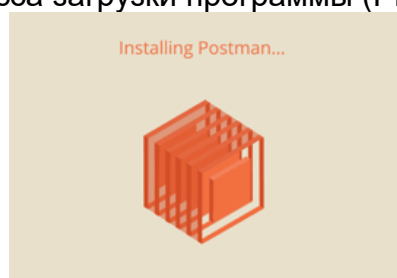


Рисунок 25

2.1.2 Настройка Postman

Postman предоставляет многооконный интерфейс с несколькими вкладками для работы с API.

При открытии программы можно выбрать один из разделов меню экрана запуска (Рисунок 26), разделы предназначены для создания:

- запросов;
- коллекций (группы сохраненных запросов);
- сред (переносимая область для переменных);
- документации (генерируется открытая или закрытая документация на основе браузера для коллекций);
- макетов серверов;
- мониторов (позволяет производить периодический запуск коллекций для проверки ее производительности и скорости реакции);
- шаблонов (позволяет производить проверку ссылок, отслеживать вопросы GitHub, следить за состоянием URL - адресов и др.);
- сетей API (каталог открытых интерфейсов API).

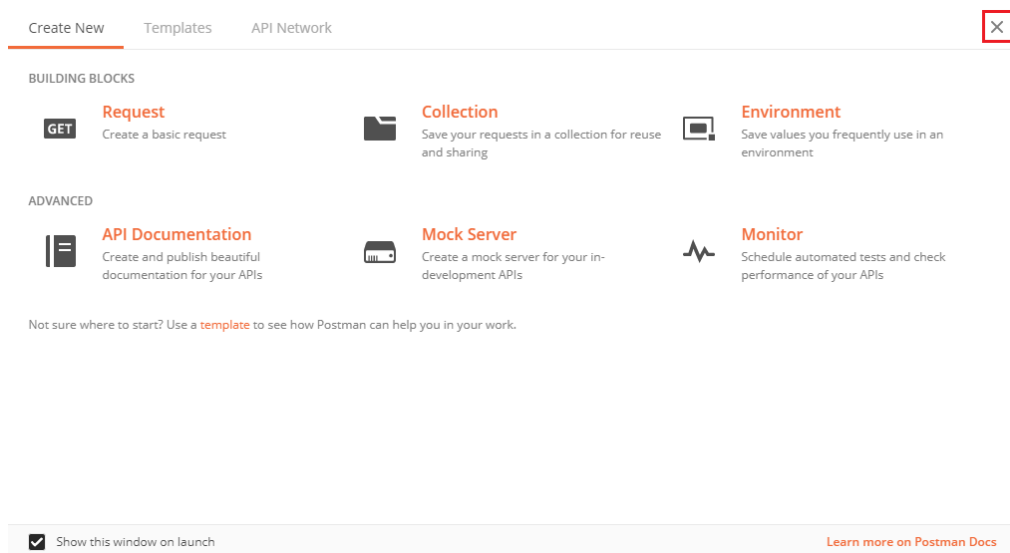


Рисунок 26 – Разделы меню программы

В случае закрытия экрана запуска открывается основная форма настроек запросов (Рисунок 27):

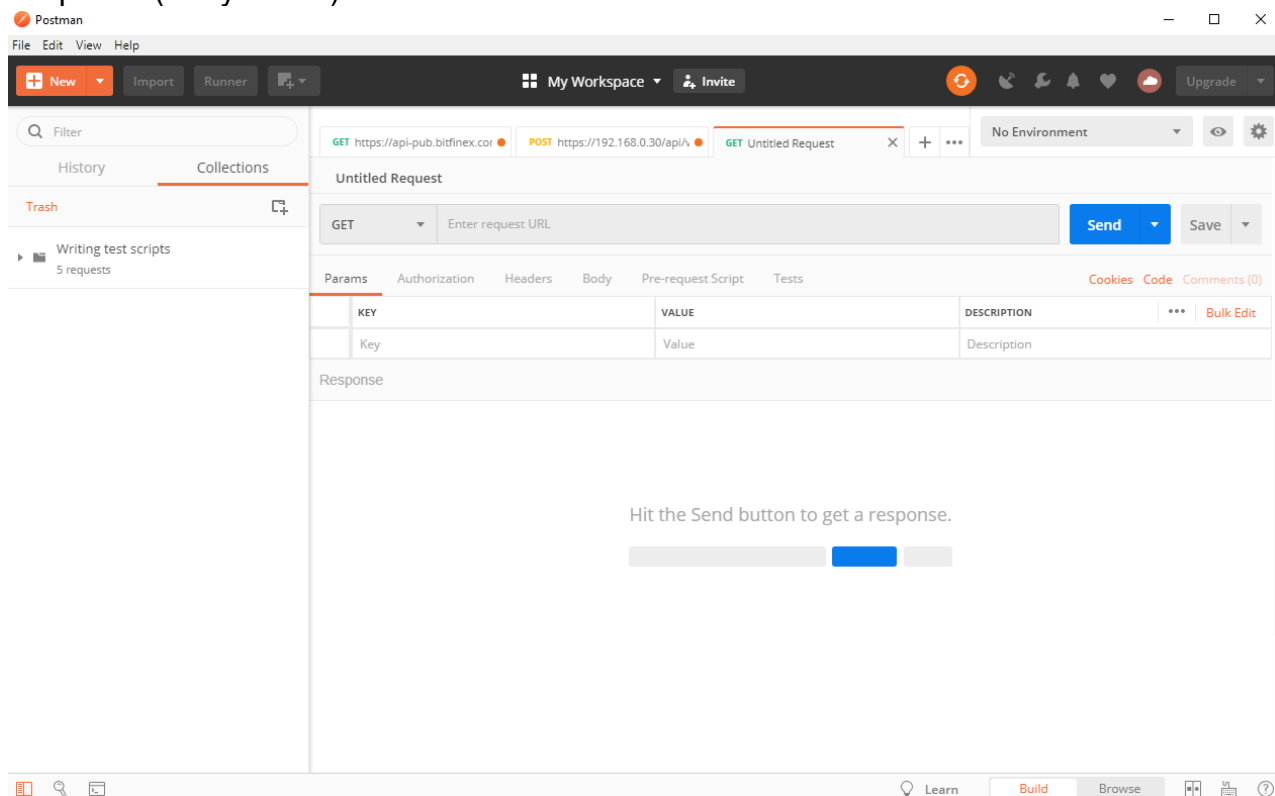


Рисунок 27 - Внешний вид формы настроек запросов

При нажатии кнопки «New» на панели инструментов открывается меню программы (Рисунок 28), список разделов аналогичен списку из меню экрана запуска (Рисунок 26).

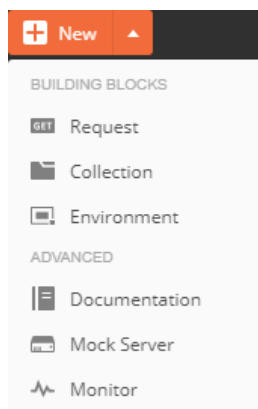


Рисунок 28 - Разделы меню программы

2.1.3 Выполнение запроса

Программа Postman позволяет выполнять HTTP - запросы, которое клиент посылает HTTP – серверу, а сервер отправляет клиенту HTTP - отклики.

Структура HTTP – запросов состоит из нескольких параметров:

- *URL - адрес* – электронный адрес, обеспечивающий доступ к веб-ресурсу в сети «интернет». URL может содержать пары «ключ-значение» или «переменную пути» (добавляются при заполнении значений на странице «Params»). Обязательный параметр для выполнения HTTP-запроса.

- *Методы HTTP* - различные виды действий, применяемые для получения отклика сервера. Наиболее распространенные методы – GET, POST, PUT и DELETE. Если в списке методов не указан интересующий вариант, то его можно ввести в поле ввода и затем выбрать в списке. В данном документе будет рассмотрен метод «POST», который позволяет добавлять данные в существующий файл или ресурс на сервере. Обязательный параметр для выполнения HTTP-запроса.

- *Headers (заголовки)* - при нажатии на закладку «Headers» отображается редактор значения ключа заголовков. Тип заголовка по умолчанию - Content-Type (за исключением заголовка для двоичного кода). Не обязательный параметр для выполнения HTTP-запроса.

- *Body («тело» запроса)* – перечень данных, которые передаются в запросе. Не обязательный параметр для выполнения HTTP-запроса.

В программе Postman доступно 4 режима кодирования «тела» запроса:

- *form-data* - это кодировка, используемая по умолчанию, имитирует заполнение формы на веб-сайте и ее отправку. Редактор форм-данных позволяет устанавливать пары «ключ-значение», возможно прикрепление файлов к ключу.

- *x-www-form-urlencoded* - кодировка аналогична параметрам URL, необходимо ввести пары «ключ-значение», прикрепление файлов к ключу невозможно.

- *raw* – кодировка позволяет создавать не редактируемый запрос, в котором изменяются только переменные окружения. Редактор «raw» позволяет установить правильный тип заголовка. Выбор настроек «XML / JSON» в типе редактора включает подсветку синтаксиса для тела запроса, а также автоматически устанавливает заголовок Content-Type.

- *binary* – кодировка, позволяющая передавать текстовые файлы или данные в двоичном формате, например, изображения, аудио или видео файлы.

В данном документе будет рассмотрен тип кодировки «raw».

Перед созданием запроса API необходимо выполнить несколько предварительных настроек:

- Ввести значение URL - адреса (Рисунок 29).

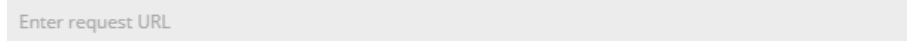


Рисунок 29 – Ввод URL - адреса

- Выбрать метод POST в списке методов, в данном методе параметры передаются не в URL, а непосредственно в теле запроса. (Рисунок 30).

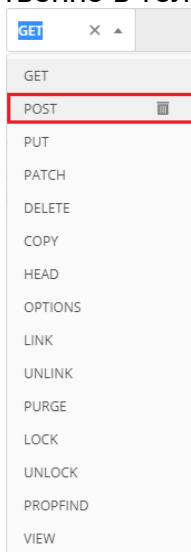


Рисунок 30 – Выбор метода запроса

- Выбрать тип параметра «Body» (Рисунок 31).

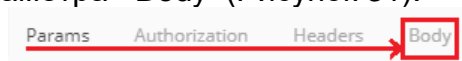


Рисунок 31 – Редактор «тело» запросов

- Указать тип кодирования «raw» (Рисунок 32).

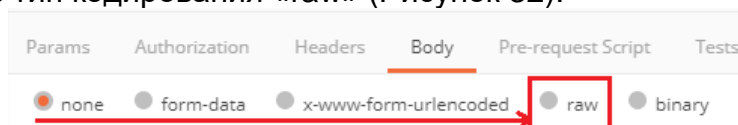


Рисунок 32 – Редактор форм данных «raw»

- Выбрать тип передачи данных в формате JSON (application/javascript) – текстовый формат обмена данными, основанный на языке программирования JavaScript. Формат хранит и передает структурированный поток информации, объекты которого представлены в виде строки. Объект JSON определяется как пара «ключ-значение» с возможностью добавлять глубокий уровень вложенности параметров (Рисунок 33).

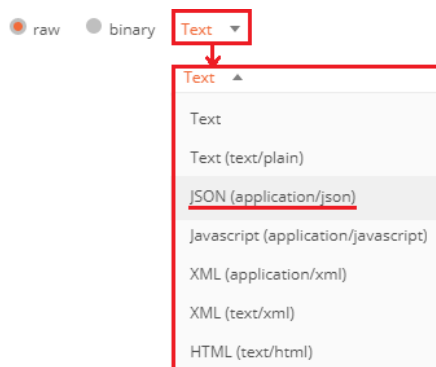


Рисунок 33 – Выбор типа передачи данных

Внешний вид формы создания запроса без данных (Рисунок 34):

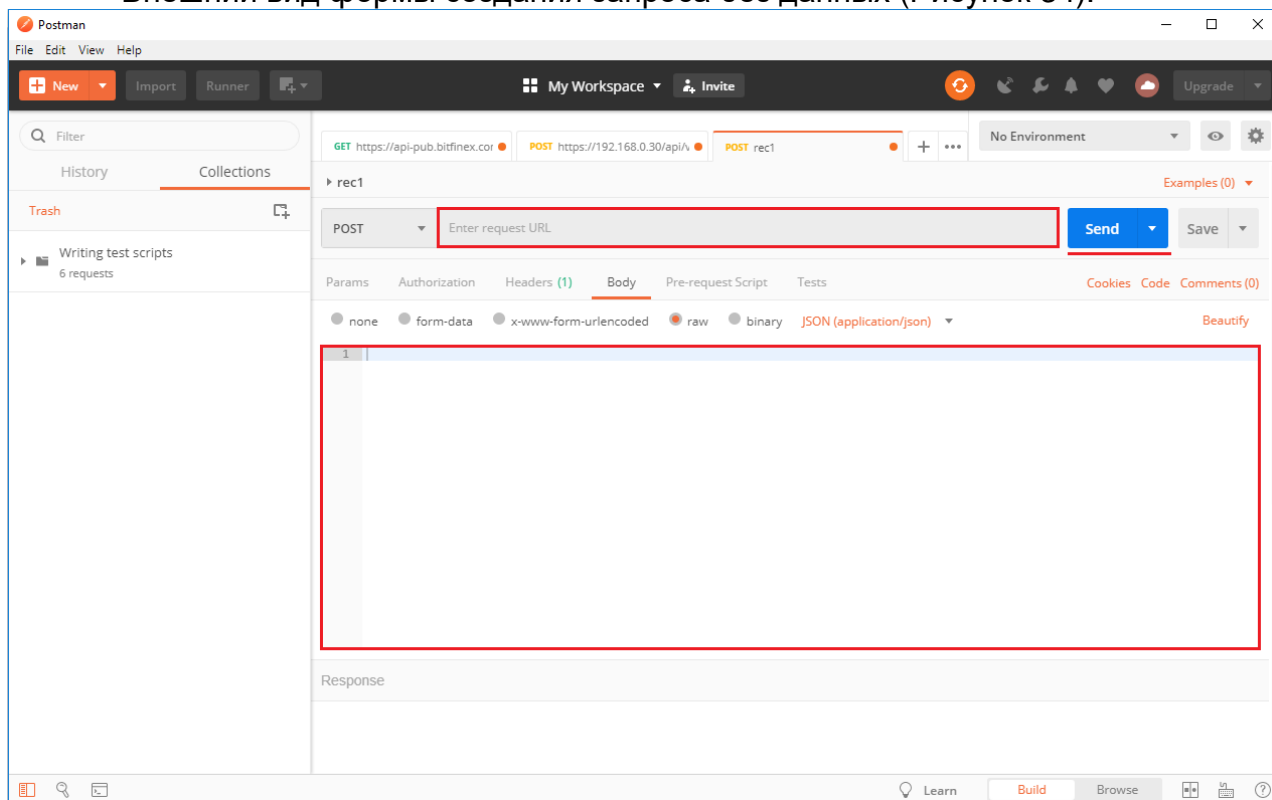


Рисунок 34 - Внешний вид формы создания запроса без данных

Для выполнения запроса API необходимо:

- 1 Ввести Url, к которому обращается запрос: «https://exs.cash/api/v1/». В тексте ссылки указан раздел – «api» и версия реализации программы – «v1».
- 2 Написать текст запроса, представленный в виде строки следующего вида:
`{"тип query": "{название метода {список ключей {список параметров}}}"}`,
 где «список параметров» является необязательным полем.

Пример запроса:

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

- 3 Нажать кнопку «Send» (Рисунок 35).

В нижней части формы получен ответ в виде списка значений параметров по каждому объекту БД.

Пример ответа:

```
{  "data": {
    "marketList": [
      {
        "name": "BTCETH",
        "stock": "BTC",
        "money": "ETH",
        "fee_prec": 4,
        "stock_prec": 8,
        "money_prec": 2,
        "min_amount": "0.00001"
      }
    ]
  }
}
```

ПРИМЕЧАНИЕ: Строка должна быть заключена в фигурные скобки («{}»), набор пар ключ-значение - в двойные кавычки («"»»).

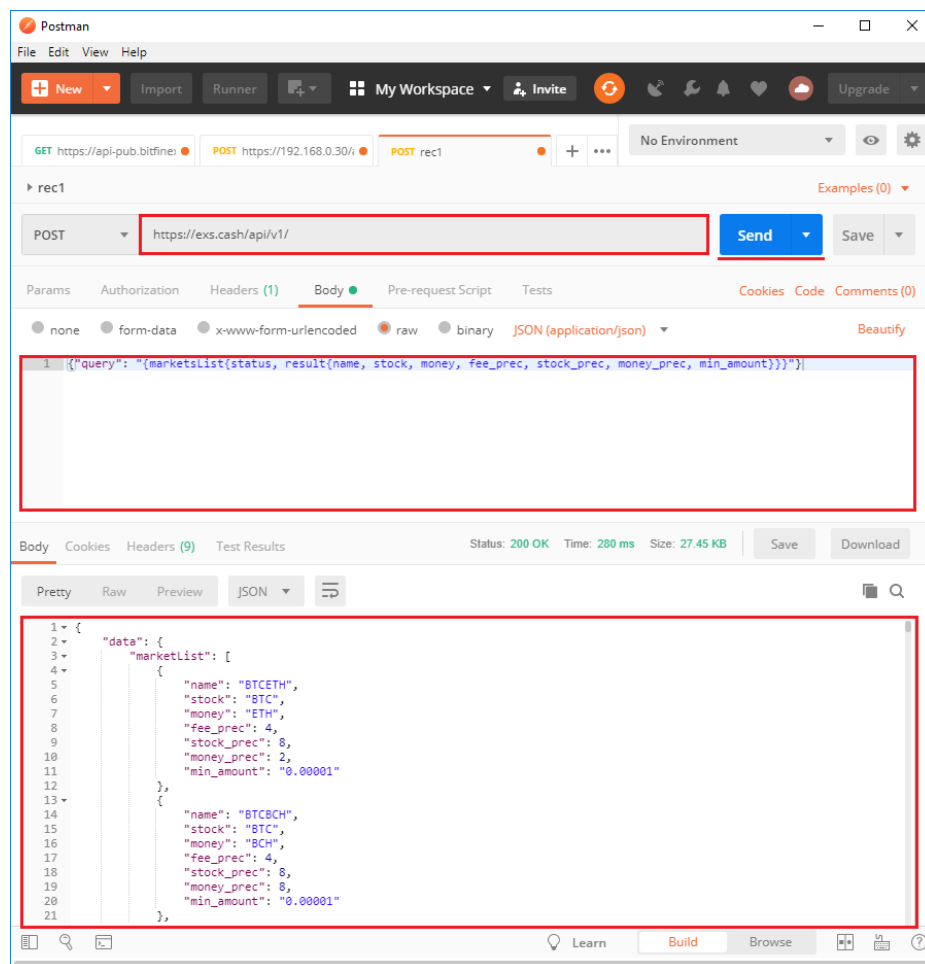


Рисунок 35 - Внешний вид формы с данными

2.2 Приложение Visual Studio Code

Visual Studio Code – это кроссплатформенный редактор с открытым исходным кодом (версии для Linux, MacOS, Windows), который осуществляет просмотр, редактирование, запуск и отладку исходного кода для приложений.

Перед установкой программы требуется провести установку Node.js.

2.2.1 Установка Node.js

Node.js – это среда разработки на языке JavaScript, которая позволяет проводить операции чтения/записи HTTP и WebSocket - запросов в API.

Установка программы включает следующие этапы:

- Перейти на страницу сайта приложения - <https://nodejs.org/en/>, открыть закладку «Downloads» и выбрать тип операционной системы (Рисунок 36). Node.js доступен как прикладная программа для Windows (32- и 64 бит), MacOS, Linux (32- и 64 бит) и смартфонов.

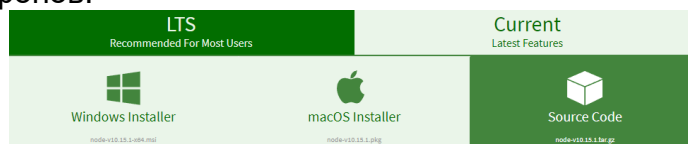


Рисунок 36 - Ссылки на установку программы на разных платформах

- Установка начинается после двойного нажатия на ярлык программы (Рисунок 37).

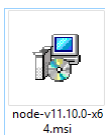


Рисунок 37 - Ярлык для установки Node.js

2.2.2 Установка Visual Studio Code

Visual Studio Code - это редактор, которая служит для создания веб-приложений и «облачных» программ.

Установка программы включает следующие этапы:

- Перейти на страницу сайта приложения - <https://code.visualstudio.com/>, в нижней части страницы выбрать тип операционной системы. Visual Studio Code доступен как прикладная программа для Windows (32- и 64 бит), MacOS и Linux (32- и 64 бит) (Рисунок 38). После выбора одного из вариантов платформы загрузка начнется автоматически. В примере показана установка для Windows 7 и более новых версий.

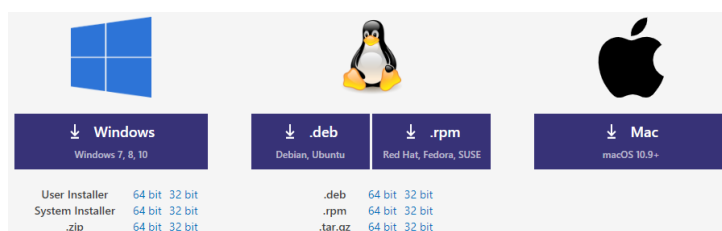


Рисунок 38 – Ссылки на установку программы на разных платформах

- Установка начинается после двойного нажатия на ярлык программы (Рисунок 39).



Рисунок 39 – Ярлык для установки Visual Studio Code

2.2.3 Настройка Visual Studio Code

В Visual Studio Code тестирование кода выполняется в основном окне редактора при вводе команд в командной строке.

Перед открытием программы необходимо создать новую папку, в которую на следующих шагах будут сохранены файлы и выгружены библиотеки.

Для открытия новой папки следует запустить программу Visual Studio Code, раскрыть меню «File» на панели инструментов, выбрать пункт «Open Folder» и указать путь до папки (Рисунок 40).

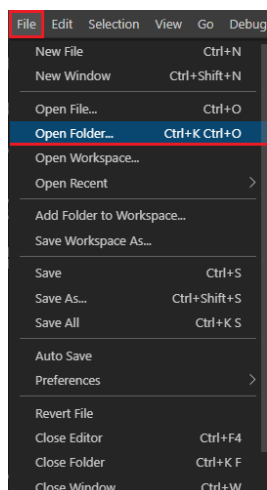


Рисунок 40 – Разделы меню программы «Visual Studio Code»

После перехода к папке открывается окно редактора программы, которое разделено на три области (Рисунок 41):

- В левой части формы (1) – проводник, панель перехода к файлам папки;
- В верхней части формы (2) – раздел отображения документов (запросов), каждый документ открывается в виде отдельной закладки;
- В нижней части формы (3) – раздел настроек формы и командная строка терминала (включает закладки «Problems», «Output», «Debug Console», «Terminal»).

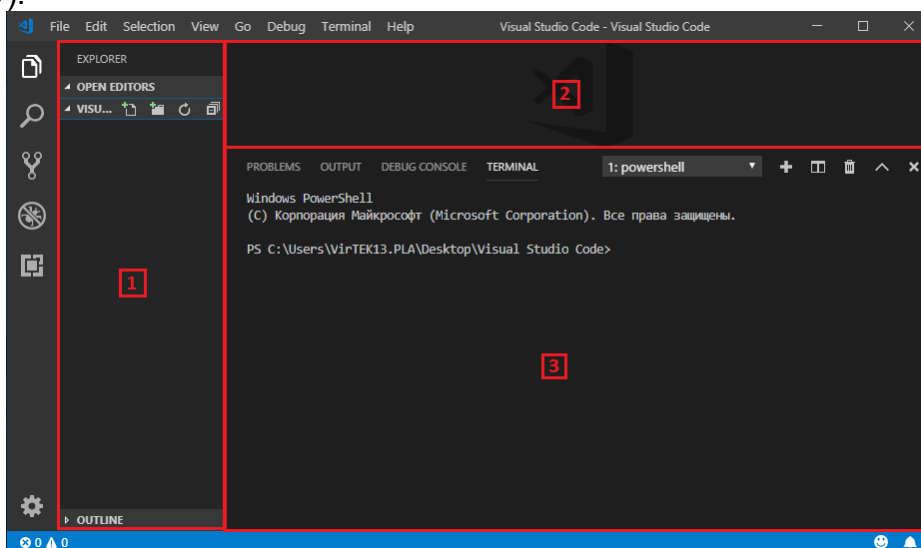


Рисунок 41 - Окно редактора программы «Visual Studio Code»

Для настройки Visual Studio Code необходимо выполнить несколько команд в командной строке, в области 3 на закладке «Terminal». По умолчанию терминал открывается в папке, выбранной в проводнике.

Список команд настройки окна редактора:

node -v – получение версии программы Node.js (Рисунок 42).

```
PS C:\Users\VirTEK13.PLA\Desktop\Visual Studio Code> node -v
v11.10.0
```

Рисунок 42 – Команда node -v

npm -v – добавление библиотеки Node.js для JavaScript (Рисунок 43).

```
PS C:\Users\VirTEK13.PLA\Desktop\Visual Studio Code> npm -v
6.7.0
```

Рисунок 43 – Команда npm -v

npm init – передача настроек по проекту (имя приложения / пакета), сведения о текущей версии приложения, краткое описание приложения, точка входа в приложение, расположение репозитория настроек для инструментальных средств, ключевые слова, информация об авторе и сведения о лицензии пакета) в создаваемый файл `package.json`. Информация записывается в файл в формате «вопрос - ответ» (Рисунок 44).

После ввода данные могут быть сохранены при нажатии клавиши «Enter» (другой вариант сохранения данных - ввести на клавиатуре символ «y»), в случае изменения данных ввести «n» и повторить команду. В приведенном примере в прямоугольнике желтого цвета показан список полей настроек до момента сохранения файла, в прямоугольнике красного цвета приведен формат данных, записанных в файле `package.json`.

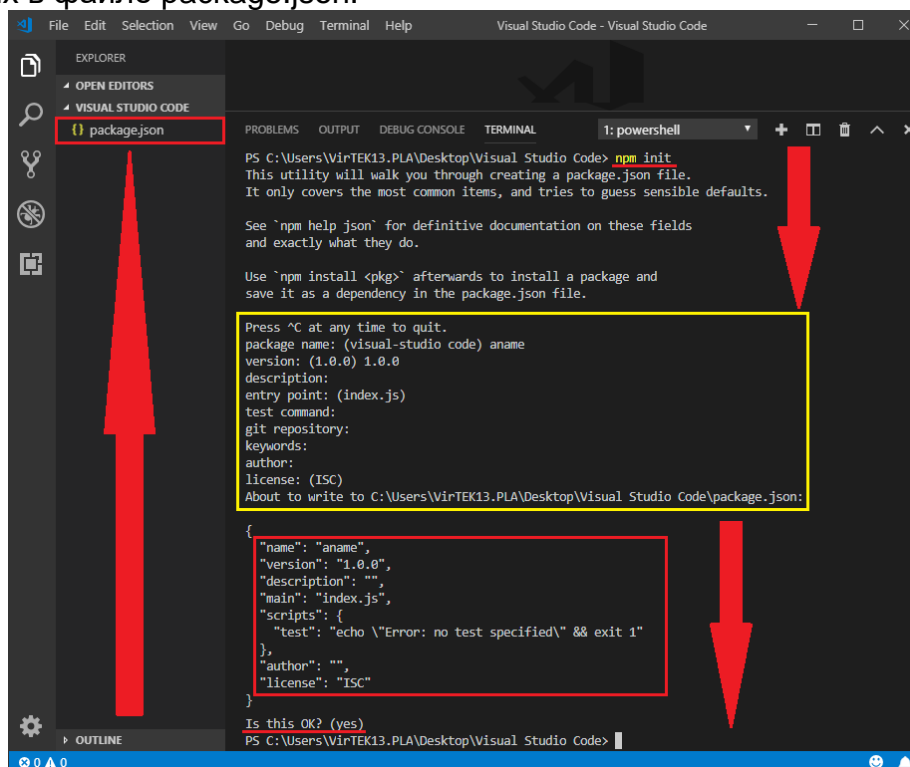


Рисунок 44 – Команда *npm init*

npm i crypto request --save – добавление библиотек, которые требуются для запроса «crypto». Флаг `--save` позволяет установить пакет и добавить запись о нём в раздел `dependencies` файла `package.json`, который описывает зависимости проекта (Рисунок 45).

Для выполнения Websocket-запросов необходимо установить дополнительные библиотеки, выполнив команду:

npm i apollo-link-ws graphql-tag subscriptions-transport-ws ws apollo-link graphql utils.

После выполнения команды будут созданы новые файлы:

- `node_modules` – папка, содержащая все модули, которые добавлены с помощью менеджера пакетов для Node.js.

- `package-lock.json` – дерево зависимостей к установленным библиотекам, хранит сведения о версии каждого установленного пакета на момент установки.

ПРИМЕЧАНИЕ: Вместо данной команды можно выполнить команду *npm install crypto request*, команда на установку библиотек «crypto» и «request».

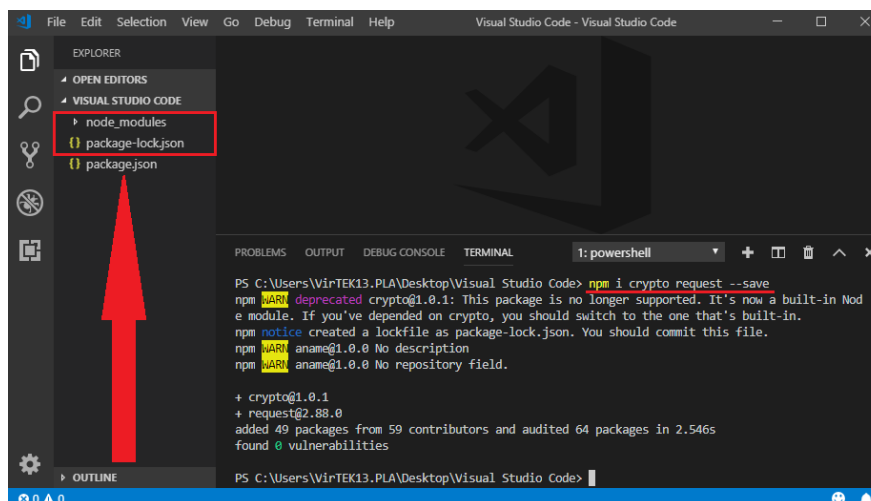


Рисунок 45 – Команда `npm i crypto request --save`

2.2.4 Выполнение запроса

Программа Visual Studio Code позволяет выполнять запросы по следующим протоколам:

- HTTP - протокол передачи гипертекста для возвращает произвольный формат данных, поддерживает структура «клиент-сервер». HTTP не сохраняет текущее состояние и настройки запроса для новых запросов.
- WebSocket - протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени. В отличие от протокола HTTP данные по протоколу WebSocket приходят, когда происходит событие, на которое производилась подписка. До закрытия соединения подписка остается активной и ожидает ответы сервера.

Перед началом работы необходимо открыть запрос в разделе отображения документов (область 2, Рисунок 41). Для этого можно скопировать его в ту же папку, в которой создан проект (Рисунок 46):

- Раскрыть меню «File» на панели инструментов, выбрать пункт «Open File» и указать путь до скрипта. Запрос откроется в разделе проводника «Open editors».
- Выделить запрос в разделе «Open editors», открыть меню «File» и, выбрав пункт «Save as», сохранить запрос (необходимо указать путь до папки, в которой создан проект).

Раскрыть запрос двойным нажатием мыши, документ откроется в виде закладки в области 2.

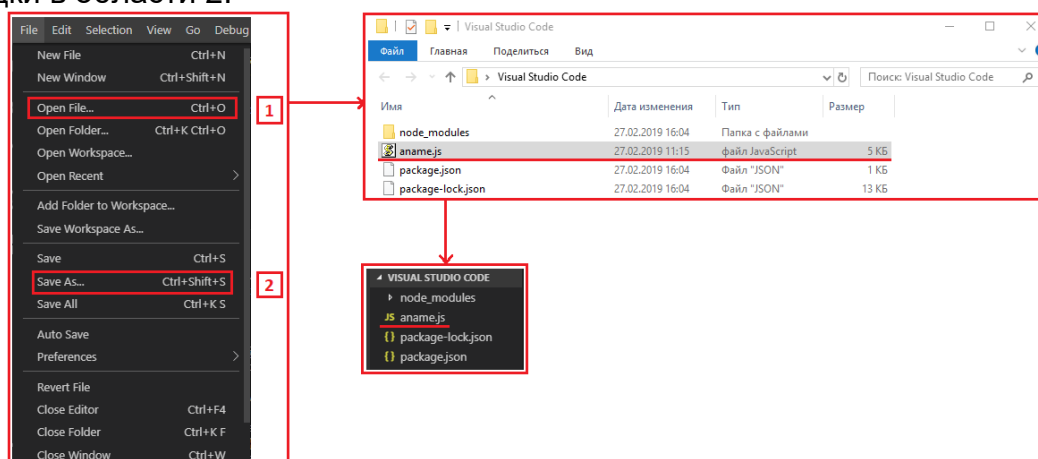


Рисунок 46 – Перенос запроса в папку

Для выполнения запроса необходимо ввести следующую команду:

node + «название файла без указания расширения» - интерпретатор файла (Рисунок 47).

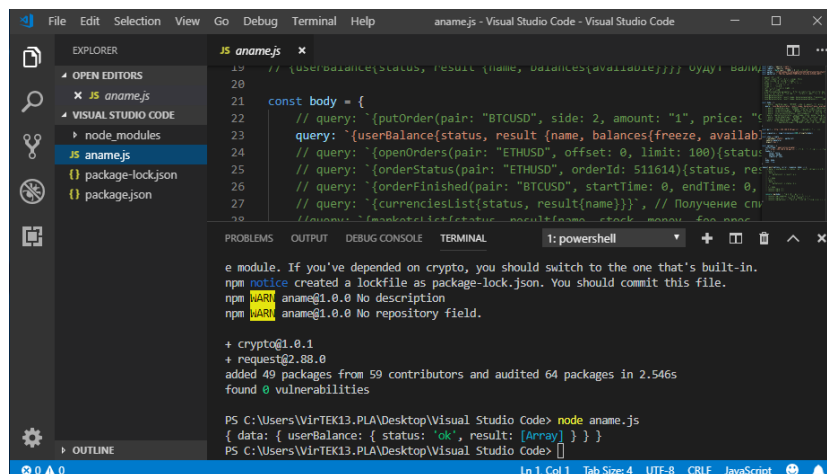


Рисунок 47 – Команда запуска скрипта

Описание работы с запросами в программе Visual Studio Code приведено в разделе 3 текущего документа (подробнее см. раздел 3). Полный текст спецификации запросов для API Биржи EXS.CASH приведен в Приложениях А и Б (подробнее см. «Приложение А. Тест HTTP-запроса для API Биржи», «Приложение Б. Тест Websocket -запроса для API Биржи»).

Внешний вид программы Visual Studio Code с текстом запроса:

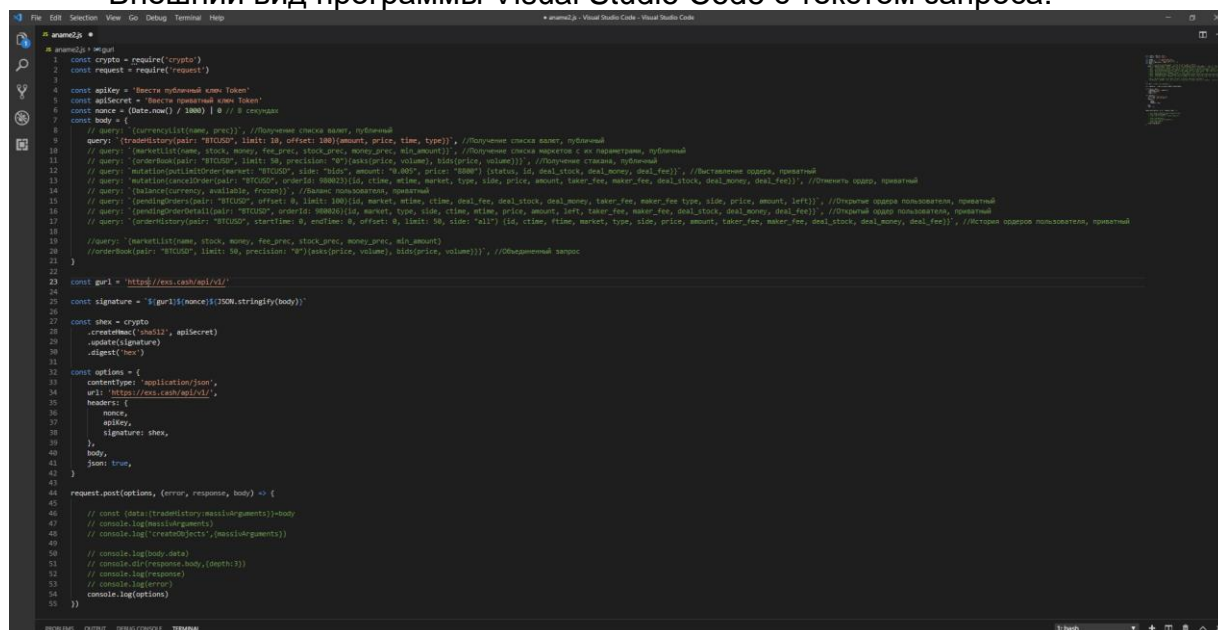


Рисунок 48 - Внешний вид формы с текстом запроса

3 Описание запроса

Запросы написаны на языке JavaScript с применением GraphQL и JSON.

GraphQL – это язык запросов для API и серверный runtime (срок исполнения программы) для выполнения запросов. Синтаксис запроса серверу передается в GraphQL, в случае невозможности вывода данных в запрошенном формате будет выведено сообщение об ошибке.

JSON – формат передачи данных запроса. API основан на JSON RPC протокола WebSocket. WebSocket - расширение HTTP, которое позволяет приложениям поддерживать многопользовательское взаимодействие в режиме реального времени, с помощью этого подхода передаются асинхронные запросы к серверу, ответы обрабатываются с помощью функций обратного вызова.

В данном разделе документации будут подробно рассмотрены составные части запросов по протоколам HTTP и WebSocket на примере выполнения скрипта в программе Visual Studio Code. Выполнение запросов по протоколу HTTP в программе Postman описано в разделе 2.1 текущей документации (подробнее см. раздел 2.1).

Перед выполнением запроса следует изменить следующие настройки:

1. Проверить и при необходимости изменить формат параметра «Дата и время» на компьютере пользователя. Настройки должны быть синхронизированы со временем сервера (тип синхронизации времени по интернету).

2. Ввести публичный и приватный ключи Token в полях «apiKey» и «apiSecret» в «теле» запроса. Получение ключей описано в разделе 1 текущей документации (подробнее см. – разделы 1.4, 1.5, 1.6).

Для выполнения запроса необходимо:

1. Для протокола HTTP закомментировать запросы, которые не будут выполняться в текущей сессии (в скрипте приведено 10 запросов в разделе про query).

Для протокола WebSocket указать выполняемый запрос в строке:

```
const request = requestTable.<Название запроса>
```

До закрытия соединения подписка остается активной и ожидает ответы сервера. Для закрытия соединения нажать сочетание клавиш «Ctrl» + «C».

2. Выбрать тип вывода данных (в разделе про console).

3. После внесения любых изменений нажать сочетание кнопок CTRL + «S» для сохранения последних изменений запроса.

4. Написать текст запроса; формат запроса зависит от типа протокола.

Структура запроса по протоколу HTTP:

{тип query: `{название метода {список ключей {список параметров}}}`}, где «список параметров» является необязательным полем.

Пример запроса по протоколу HTTP:

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

Структура запроса по протоколу WebSocket:

{тип query: `subscription + название метода (объявление типа переменной: строка) {список ключей (ссылка на тип переменной) {список параметров}}`, тип переменной: {название валюты или валютной пары}}. «Список параметров» является необязательным полем.

Пример запроса по протоколу WebSocket:

```
query: `
```

```

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

```

5. Выполнить запрос позволяет команда *node* + «название файла без указания расширения» (в примере, *node aname*).

ПРИМЕЧАНИЕ: В запросах набор пар «ключ-значение» может быть заключен в одинарные кавычки двух типов – «`» или «' ». Значения типа «текст» в скрипты передаются в двойных кавычках – « », значения типа «число» или «boolean» передаются без кавычек (например, тип «текст» - «BTCUSD» и тип «число» - 1).

3.1 Структура запроса

Спецификация запроса содержит несколько разделов, каждый из которых имеет собственное функциональное предназначение:

1) Передача констант. Раздел позволяет определить формат ввода и вывода информации.

2) Передача запроса *query*. Раздел определяет метод передачи данных и набор списка ключей и их значений для вывода информации в заданном формате.

3) Передача данных и вывод информации в веб-консоль. Раздел определяет формат вывода информации в ответ на выполнение запроса сервером.

Подробное описание каждого раздела приведено ниже.

1) Передача констант.

Ключевое слово *const* обозначает константу - спецификатор, который изменяет поведение переменной и делает ее доступной только для чтения. При попытке присвоить иное значение константе, компилятор выдает ошибку.

Список констант для запросов HTTP и Websocket приведен в Таблица 1.

Таблица 1 – Список констант для запросов HTTP и Websocket

№	Константы переменной HTTP	Константы переменной Websocket
1	«crypto» Библиотека предоставляет функционал шифрования (включает набор для хеширования, шифраторов, дешифраторов и функций верификации). <pre>const crypto = require('crypto')</pre>	
2	«apiKey» и «apiSecret» Настройка пары публичного-приватного ключей Token пользователя (закладка «Личные данные» на странице https://exs.cash/settings , подробнее см. раздел 1). Для передачи данных запроса необходимо ввести значение ключей в одинарных кавычках («'» в полях «Ввести публичный ключ Token» и «Ввести приватный ключ Token». В случае, если ключи Token (публичный и приватный) не указаны, невозможно выполнить запросы типа «приватный метод», на выполнение запросов типа «публичный метод» ввод ключей не влияет. <pre>const apiKey = 'Ввести публичный ключ Token'</pre> <pre>const apiSecret = 'Ввести приватный ключ Token'</pre>	
3	«nonce» Формат передачи времени в Unix, значение рассчитывается в секундах. Производится расчет текущего времени на компьютере пользователя в секундах, при отправлении запроса происходит проверка совпадения времени на компьютере пользователя и на сервере. В случае совпадения запрос выполняется, в случае расхождения будет возвращена ошибка о невозможности выполнения запроса. <pre>const nonce = (Date.now() / 1000) 0</pre>	
4	«request» Библиотека передает запросы через JavaScript. <pre>const request = require('request')</pre>	«request» Библиотека передает название выполняемого метода. Список методов приведен в разделе 3.2. <pre>const request = requestTable.<Название метода></pre>
5	«gurl» Формат передачи адреса глобального url. В тексте ссылки указан раздел – «api» и версия реализации программы – «v1». <pre>const gurl = `https://exs.cash/api/v1/`</pre>	«wsUrl» и «httpUrl» Формат передачи адреса глобального url: - wsUrl - подключение протокола Websocket; - httpUrl - добавление подписи (signature). В тексте ссылки указан раздел – «api» и версия реализации программы – «v1». <pre>const wsUrl = 'ws://192.168.0.30/api/v1'</pre> <pre>const httpUrl = 'http://192.168.0.30/api/v1/'</pre>
6	«signature» Формат записи подписи, который включает: - проверку url; - совпадения времени, установленного на компьютере пользователя со временем на сервере;	«signature» Процесс создания формата и объявления подписи, который включает: - проверку url; - совпадения времени, установленного на компьютере пользователя со временем на сервере;

№	Константы переменной HTTP	Константы переменной Websocket
	<p>- метод JSON.stringify(), который преобразует значение JavaScript в строку JSON. В данном случае происходит подстановка значений в «тело» запроса.</p> <p>Область видимости переменной – блоки значений, переданные в фигурных скобках – «{ }», и объединенные символом «\$». Данный тип переменной становится видимым только после объявления, при использовании в цикле для каждой итерации создается своя переменная.</p> <pre>const signature =` \${gurl}\${nonce}\${JSON.stringify(body)}`</pre>	<p>- метод JSON.stringify(), который преобразует значение JavaScript в строку JSON. В данном случае происходит подстановка значений в «тело» запроса.</p> <p>- приватный ключ Token;</p> <p>- библиотеку шифрования «crypto» использует пакет createHmac, шифруется ключами «sha512» и секретным ключом Token, происходит обновление подписи и обработка информации методом hex.</p> <p>- HMAC - один из механизмов проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадежной среде, не были изменены сторонними лицами.</p> <p>- ключ «sha512» - алгоритм хеширования, который является функцией криптографического алгоритма SHA-2. Применяется в различных приложениях, связанных с защитой информации.</p> <p>- Hex - 16-ричная системы исчисления (16-битная адресация).</p> <p>Данные помещаются в Header, body, происходит отправление запроса.</p>
7	<p>«shex»</p> <p>Процесс преобразования данных включает следующие элементы:</p> <p>- Библиотека шифрования «crypto» использует пакет createHmac, шифруется ключами «sha512» и секретным ключом Token, происходит обновление подписи и обработка информации методом hex.</p> <p>- HMAC - один из механизмов проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадежной среде, не были изменены сторонними лицами.</p> <p>- Ключ «sha512» - алгоритм хеширования, который является функцией криптографического алгоритма SHA-2. Применяется в различных приложениях, связанных с защитой информации.</p> <p>- Hex - 16-ричная системы исчисления (16-битная адресация).</p> <p>Данные помещаются в Header, body, происходит отправление запроса.</p> <pre>const shex = crypto .createHmac('sha512', apiSecret) .update(signature) .digest('hex')</pre>	<pre>const getServerSignature = ({ nonce, body, secretKey }) => { const data = `\${httpUrl}\${nonce}\${JSON.stringify(body)}` const serverSignature = crypto .createHmac('sha512', secretKey) .update(data) .digest('hex') return serverSignature } const signature = getServerSignature({ nonce, body: request.query, secretKey, })</pre>
8	<p>«options»</p> <p>Набор параметров, которые добавляются к запросу:</p> <p>- содержимое HTTP - запроса зашифровано как строка query;</p> <p>- ссылка на адрес url;</p>	<p>«options»</p> <p>Формат вывода набора параметров, которые добавляются к запросу:</p> <p>- заголовок запроса (формат передачи времени в Unix, публичный ключ Token, подпись);</p>

№	Константы переменной HTTP	Константы переменной Websocket
	<p>- содержание запроса передается в "тело" запроса. В заголовок выносятся следующие показатели: текущая дата и время, публичный и приватный ключи от Token, подпись в формате, заданном при определении константы shex.</p> <p>- информация о типе передачи данных (в формате JSON).</p> <pre>const options = { contentType: 'application/ json', url: `https://exs.cash/api/v1/`, headers: { nonce, apiKey, signature: shex, }, body, json: true, }</pre>	<p>- результаты выполнения запроса.</p> <pre>const options = { headers: { nonce, apikey, signature }, body: request.query, }</pre>
9	-	<p>«WebSocketLink» (функция из пакета «apollo-link-ws») Библиотека для создания GraphQL-клиента, который работает по протоколу Websocket.</p> <pre>const { WebSocketLink } = require('apollo-link-ws')</pre>
10	-	<p>«gql» Библиотека предназначена для анализа GraphQL-запроса (парсинг) при определении настроек конфигурационного файла. Дополнительно проверить подключение библиотеки, которая содержит программное обеспечение для graphql-tag – GraphQL.</p> <pre>const gql = require('graphql-tag').default</pre>
11	-	<p>«SubscriptionClient» (функция из пакета «subscriptions-transport-ws») Библиотека для создания клиента, который выполняет подготовку и передачу информации, полученной с сервера, по подпискам протокола Websocket.</p> <pre>const { SubscriptionClient } = require('subscriptions-transport-ws')</pre>
12	-	<p>«ws» Библиотека для создания пакета Websocket-соединения.</p> <pre>const ws = require('ws')</pre>
13	-	« execute » (функция из пакета «apollo-link»)

№	Константы переменной HTTP	Константы переменной Websocket
		Библиотека для выполнения функций запроса. <code>const { execute } = require('apollo-link')</code>
14	-	«utils» Библиотека содержит пакет вспомогательных утилит для Node. <code>const util = require("util")</code>
15	-	«requestTable» Структура содержит список методов, которые подробно описаны в разделе 3.2.
16	-	«createSubscriptionObservable» и «subscribe» Процесс создания формата и объявления запуска выполнения протокола наблюдения за событиями (подписок) включает: - обращение к библиотеке WebSocketLink; - обращение к библиотеке SubscriptionClient; - подключение протокола Websocket wsUrl; - передачу параметров. Производится подготовка клиента для отправления GraphQL-запроса серверу и выполнение запроса. В отличие от протокола HTTP данные по протоколу WebSocket поступают при наступлении события, на которое производилась подписка. До закрытия соединения подписка остается активной и ожидает ответы сервера. <pre>const createSubscriptionObservable = (wsUrl, query, variables) => { const link = new WebSocketLink(new SubscriptionClient(wsUrl, { connectionParams: options, }, ws)) return execute(link, { query,</pre>

№	Константы переменной HTTP	Константы переменной Websocket
		<pre> variables, }) } const subscribe = () => { const subscriptionObservable = createSubscriptionObservable(wsUrl, gql` \${request.query} `, request.variables) subscribe() } </pre>

2) Передача запроса query в виде пары «ключ-значение».

В запросе query передается список методов, позволяющих проводить операции по API Биржи. В документе описано 10 методов HTTP и 7 методов Websocket (подробнее см. раздел 3.2).

Активным должен быть только выполняемый HTTP-запрос, все остальные запросы закомментированы. Для выполнения нескольких методов одновременно запросы необходимо передать в один query.

Пример объединения двух запросов в один:

- запрос 1;

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

- запрос 2;

```
query: `{orderBook(pair: "BTCUSD", limit: 50, precision: "0"){asks{price, volume}, bids{price, volume}}}`,
```

- запрос 3, объединяющий запросы 1 и 2 (для объединения запросов между ними удалены следующие символы: «}`, query: `{»).

```
query: `{marketList{name, stock, money, fee_prec, stock_prec, money_prec, min_amount}
      orderBook(pair: "BTCUSD", limit: 50, precision: "0"){asks{price, volume}, bids{price, volume}}}`,
```

Для Websocket-запросов создана структура requestTable, которая включает список из всех запросов, название выполняемого запроса должно передаваться в переменной request.

```
const request = requestTable.<Название метода>
```

Параметры для получения значений могут быть указаны в любом порядке, количество параметров не ограничено, примеры записей:

```
tradeHistory(pair: "BTCUSD", limit: 10, offset: 100){price, time}
tradeHistory(pair: "BTCUSD", limit: 10, offset: 100){amount, price, time, type}
```

3) Вывод информации в веб-консоль.

При передаче данных запроса формата POST происходит обращение Node.js к стороннему API и получения ответа. Форма ответа - сообщение об ошибке или текст ответа. В примере приведен формат передачи данных для HTTP-запроса.

```
request.post(options, (error, response, body) => {«Укажите форму вывода данных»})
```

ПРИМЕЧАНИЕ. Форма вывода информации заключается в фигурные скобки («{ }»).

Формат вывода ответа приходит в виде сообщения в веб - консоль с параметрами согласно формату запроса (возможные типы вывода информации в console):

1) Вывод перечня параметров в виде массива и создание нового объекта, содержащего данные массива.

```
const {data:{tradeHistory:massivArguments}}=body
console.log(massivArguments)
console.log('createObjects',{massivArguments})
```

2) Вывод полного списка параметров запроса с присвоенными значениями.

```
console.log(body.data)
```

3) Вывод структуры переданного объекта с раскрытием данных запроса до указанного уровня глубины.

```
console.dir(response.body,{depth:3})
```

4) Передача настроек сервера при выполнении запроса.

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

5) Получение сообщения об ошибке. В случае успешного выполнения запроса выводится ответ "null".

```
console.log(error)
```

6) Получение списка переданных в запросе параметров и текста запроса.

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

7) Информационное сообщение о том, что подписка на метод произведена, лог в состоянии ожидания событий.

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

8) Вывод полного списка параметров запроса с присвоенными значениями и добавлением глубины раскрытия уровней; информация получена с использованием библиотеки utils от Node.

```
console.log('Received event: ', util.inspect(eventData, { depth: 10 })))
```

9) Вывод полного списка параметров запроса с присвоенными значениями и добавлением глубины раскрытия уровней; информация получена с использованием стандартной функцией Java-скрипт. Формат вывода предыдущего и текущего лога аналогичен.

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

Форматы вывода информации проиллюстрированы в разделе «Выходные данные» (подробнее см. раздел 3.3).

3.2 Описание методов

Для настройки API Биржи применяются 10 методов HTTP (Таблицы 3-12) и 7 методов Websocket (Таблицы 13-19), подробнее см. Таблица 2.

Таблица 2 – Список методов HTTP и Websocket:

HTTP		Websocket	
публичные методы	приватные методы	публичные методы	приватные методы
currencyList (Таблица 3)	putLimitOrder (Таблица 7)	marketUpdated (Таблица 13)	balanceUpdated (Таблица 18)
tradeHistory (Таблица 4)	cancelOrder (Таблица 8)	stateUpdated (Таблица 14)	orderUpdated (Таблица 19)
marketList (Таблица 5)	balance (Таблица 9)	chartUpdated (Таблица 15)	
orderBook (Таблица 6)	pendingOrders (Таблица 10)	orderBookUpdated (Таблица 16)	
	pendingOrderDetail (Таблица 11)	dealsUpdated (Таблица 17)	
	orderHistory (Таблица 12)		

Структура передачи методов HTTP:

```
const body = {query: `{Передать запрос и его параметры}`, }.
```

Структура передачи методов Websocket:

```
const requestTable = {marketUpdated: {  
  query: `{ Передать запрос}`,  
  variables: { Передать параметры запроса}, }, }  
const request = requestTable.<Название метода>
```

Методы включают текст запроса и его параметры и передаются в формате «ключ – значение». Параметры запроса являются необязательным атрибутом.

Для HTTP может быть выполнен только один запрос, остальные запросы должны быть закомментированы. Пример объединения нескольких запросов в один приведен в абзаце «Передача запроса query в виде пары «ключ-значение» (подробнее см. раздел 3.1).

Для Websocket в константе «requestTable» приведен полный список методов, константа «request» содержит название вызываемого метода.

ПРИМЕЧАНИЕ. Текст запроса и его параметры заключаются в фигурные скобки («{ }»).

Таблица 3 – Метод currencyList (Получение списка валют)

Наименование метода	currencyList
Описание метода	Получение списка валют
Тип протокола	HTTP
Тип метода	Публичный метод
Входящие параметры	—
Исходящие параметры	name - название валюты Биржи (криптовалюты и фиатные деньги); prec - количество знаков после запятой у видов валют.
Пример использования	<code>query: `{currencyList{name, prec}}`,</code>
Пример ответа	<pre>{ currencyList: [{ name: 'BTC', prec: 8 }, { name: 'ETH', prec: 8 }, { name: 'BCH', prec: 8 }, { name: 'LTC', prec: 8 }, { name: 'ADA', prec: 8 }, { name: 'XMR', prec: 8 }, { name: 'USDT', prec: 8 }, { name: 'EXS', prec: 8 }, { name: 'XEM', prec: 8 }, { name: 'DASH', prec: 8 }, { name: 'ZEC', prec: 8 }, { name: 'BTG', prec: 8 }, { name: 'DCR', prec: 8 }, { name: 'SC', prec: 8 }, { name: 'WAVES', prec: 8 }, { name: 'STORJ', prec: 8 }, { name: 'NMC', prec: 8 }, { name: 'PLT', prec: 8 }, { name: 'EUR', prec: 2 }, { name: 'USD', prec: 2 }, { name: 'RUB', prec: 2 }] }</pre>

Таблица 4 – Метод tradeHistory (Получение истории сделок)

Наименование метода	tradeHistory
Описание метода	Получение истории сделок
Тип протокола	HTTP

Тип метода	Публичный метод
Входящие параметры	pair - название пары валют; limit - количество отображаемых позиций (по умолчанию 10); offset - количество позиций, которые нужно пропустить от начала списка (по умолчанию 100).
Исходящие параметры	amount - количество выставленных единиц валюты; price - цена за единицу валюты при покупке/продаже ордера; time - формат передачи времени в Unix; type - тип операции («1» / «sell» - продажа или «2» / «buy» - покупка).
Пример использования	<pre>query: `{tradeHistory(pair: "BTCUSD", limit: 10, offset: 100){amount, price, time, type}}`</pre>
Пример ответа	<pre>{ tradeHistory: [{ amount: '25', price: '1', time: '1560765457.677826', type: 'sell' }, { amount: '200', price: '0.02', time: '1560765411.456113', type: 'buy' }] }</pre>

Таблица 5 – Метод marketList (Получение списка маркетов и их параметров)

Наименование метода	marketList
Описание метода	Получение списка маркетов и их параметров
Тип протокола	HTTP
Тип метода	Публичный метод
Входящие параметры	—
Исходящие параметры	name - название валюты Биржи (криптовалюты и фиатные деньги); stock - приобретаемый вид валюты; money - денежный эквивалент; fee_prec - количество знаков после запятой для типа «fee» (комиссия); stock_prec - количество знаков после запятой у валюты типа «stock»; money_prec - количество знаков после запятой у валюты типа «money»; min_amount - минимальный объем для совершения операции по приобретению валюты.
Пример использования	<pre>query: `{marketList{name, stock, money, fee_prec, stock_prec, money_prec, min_amount}}`</pre>
Пример ответа	<pre>{ marketList: [{ name: 'BTCETH', stock: 'BTC', money: 'ETH', fee_prec: 4, stock_prec: 8, money_prec: 2, min_amount: '0.00001' }, { name: 'BTCBCH', stock: 'BTC', money: 'BCH', fee_prec: 4,</pre>

	<pre>stock_prec: 8, money_prec: 8, min_amount: '0.00001' }] }</pre>
--	--

Таблица 6 – Метод orderBook (Получение списка ордеров по продаже/ покупке)

Наименование метода	orderBook
Описание метода	Получение списка ордеров по продаже/ покупке
Тип протокола	HTTP
Тип метода	Публичный метод
Входящие параметры	pair - название пары валют; limit - количество отображаемых позиций (по умолчанию 50); precision - шаг; объединение близких по цене предложений в одну позицию.
Исходящие параметры	asks - список ордеров на продажу (предложение); bids - список ордеров на покупку (спрос) price - цена за единицу валюты при покупке/продаже ордера; volume - объем; количество выставленных единиц.
Пример использования	<pre>query: `{orderBook(pair: "BTCUSD", limit: 50, precision: "0"){asks{price, volume}, bids{price, volume}}}`,</pre>
Пример ответа	<pre>{ data: { orderBook: { asks: [{ price: '1325', volume: '0.965' }], bids: [{ price: '1323', volume: '1' }] } }</pre>

Таблица 7 – Метод putLimitOrder (Выставление ордера)

Наименование метода	putLimitOrder
Описание метода	Выставление ордера
Тип протокола	HTTP
Тип метода	Приватный метод
Входящие параметры	market - название рынка; side - статус ордера (лимитный ордер - 0, ордер на покупку - 1, ордер на продажу - 2); amount - количество выставленных единиц валюты; price - цена за единицу валюты при покупке/продаже ордера.
Исходящие параметры	status - статус операции (выставленный, исполненный, частично исполненный, не исполнен в связи с недостаточностью средств); id - идентификатор ордера; deal_stock - объем по сделке при покупке ордеров; deal_money - объем по сделке при продаже ордеров; deal_fee - комиссия по сделке.
Пример использования	<pre>query: `mutation{putLimitOrder(market: "BTCUSD", side: "bids", amount: "0.005", price: "8800") {status, id, deal_stock, deal_money, deal_fee}}`,</pre>
Пример ответа	<pre>{ data: { putLimitOrder:</pre>

	<pre>{ status: 'completed', id: 980028, deal_stock: '0.005', deal_money: '6.625', deal_fee: '0.00001' } } }</pre>
--	---

Таблица 8 – Метод cancelOrder (Отмена выставленного ордера)

Наименование метода	cancelOrder
Описание метода	Отмена выставленного ордера
Тип протокола	HTTP
Тип метода	Приватный метод
Входящие параметры	pair - название пары валют; orderId – порядковый номер ордера.
Исходящие параметры	id - идентификатор ордера; ctime - время создания ордера; mtime - время последнего изменения по ордеру; market - название рынка; type - тип операции («1» / «sell» - продажа или «2» / «buy» - покупка); side - статус ордера (лимитный ордер - 0, ордер на покупку - 1, ордер на продажу - 2); price - цена за единицу валюты при покупке/продаже ордера; amount - количество выставленных единиц валюты; taker_fee - комиссия взимается в тот момент, когда пользователь выкупает выставленное предложение другого пользователя; maker_fee - комиссия взимается в тот момент, когда пользователь выставляет новое предложение о покупке / продаже ордера; deal_stock - объем по сделке при покупке ордеров; deal_money - объем по сделке при продаже ордеров; deal_fee - комиссия по сделке.
Пример использования	<pre>query: 'mutation{cancelOrder(pair: "BTCUSD", orderId: 980023){id, ctime, mtime, market, type, side, price, amount, taker_fee, maker_fee, deal_stock, deal_money, deal_fee}}',</pre>
Пример ответа	<pre>{ data: { cancelOrder: { id: 980023, ctime: 1561114862.82105, mtime: 1561114862.82105, market: 'BTCUSD', type: 1, side: 2, price: 1323, amount: '1', taker_fee: 0.002, maker_fee: 0.002, deal_stock: 0, deal_money: 0, deal_fee: 0 } } }</pre>

Таблица 9 – Метод balance (Баланс пользователя)

Наименование метода	balance
Описание метода	Баланс пользователя

Тип протокола	HTTP
Тип метода	Приватный метод
Входящие параметры	—
Исходящие параметры	currency - название валюты, входящей на рынок; available - доступный баланс пользователя; frozen - зарезервированный баланс по выставленным ордерам.
Пример использования	<code>query: `{balance{currency, available, frozen}}`</code>
Пример ответа	<pre>{ data: { balance: [{ currency: 'BTC', available: '10008.98992998', frozen: '0.95' }, { currency: 'ETH', available: '9990.99', frozen: '0' }, { currency: 'BCH', available: '0', frozen: '0' }, { currency: 'LTC', available: '0', frozen: '0' }, { currency: 'ADA', available: '0', frozen: '0' }, { currency: 'XMR', available: '10000', frozen: '0' }, { currency: 'USDT', available: '0', frozen: '0' }, { currency: 'EXS', available: '0', frozen: '0' }, { currency: 'XEM', available: '0', frozen: '0' }, { currency: 'DASH', available: '0', frozen: '0' }, { currency: 'ZEC', available: '0', frozen: '0' }, { currency: 'BTG', available: '0', frozen: '0' }, { currency: 'DCR', available: '0', frozen: '0' }, { currency: 'SC', available: '0', frozen: '0' }, { currency: 'WAVES', available: '0', frozen: '0' }, { currency: 'STORJ', available: '0', frozen: '0' }, { currency: 'NMC', available: '0', frozen: '0' }, { currency: 'PLT', available: '0', frozen: '0' }, { currency: 'EUR', available: '10000', frozen: '0' }, { currency: 'USD', available: '12199.82', frozen: '0' }, { currency: 'RUB', available: '10000', frozen: '0' }] } }</pre>

Таблица 10 – Метод pendingOrders (Открытые ордера пользователя)

Наименование метода	pendingOrders
Описание метода	Открытые ордера пользователя
Тип протокола	HTTP
Тип метода	Приватный метод
Входящие параметры	pair - название пары валют; offset - количество позиций, которые нужно пропустить от начала списка (по умолчанию 0); limit - количество отображаемых позиций (по умолчанию 100).
Исходящие параметры	id - идентификатор ордера; market - название рынка; mtime - время последнего изменения по ордеру; ctime - время создания ордера; deal_fee - комиссия по сделке; deal_stock - объем по сделке при покупке ордеров; deal_money - объем по сделке при продаже ордеров; taker_fee - комиссия взимается в тот момент, когда пользователь выкупает выставленное предложение другого пользователя;

	maker_fee - комиссия взимается в тот момент, когда пользователь выставляет новое предложение о покупке / продаже ордера; type - тип операции («1» / «sell» - продажа или «2» / «buy» - покупка); side - статус ордера (лимитный ордер - 0, ордер на покупку - 1, ордер на продажу - 2); price - цена за единицу валюты при покупке/продаже ордера; amount - количество выставленных единиц валюты; left - не выкупленный остаток по ордеру относительно общего количества.
Пример использования	<pre>query: `{pendingOrders(pair: "BTCUSD", offset: 0, limit: 100){id, market, mtime, ctime, deal_fee, deal_stock, deal_money, taker_fee, maker_fee type, side, price, amount, left}}`,</pre>
Пример ответа	<pre>{ data: { pendingOrders: [{ id: 980026, market: 'BTCUSD', mtime: 1561122032.637261, ctime: 1561114877.058889, deal_fee: 0.1325, deal_stock: 0.05, deal_money: 66.25, taker_fee: 0.002, maker_fee: 0.002, type: 1, side: 1, price: 1325, amount: '1', left: '0.95' }] } }</pre>

Таблица 11 – Метод pendingOrderDetail (Открытый ордер пользователя)

Наименование метода	pendingOrderDetail
Описание метода	Открытый ордер пользователя
Тип протокола	HTTP
Тип метода	Приватный метод
Входящие параметры	pair - название пары валют; orderId - порядковый номер ордера.
Исходящие параметры	id - идентификатор ордера; market - название рынка; type - тип операции («1» / «sell» - продажа или «2» / «buy» - покупка); side - статус ордера (лимитный ордер - 0, ордер на покупку - 1, ордер на продажу - 2); ctime - время создания ордера; mtime - время последнего изменения по ордеру; price - цена за единицу валюты при покупке/продаже ордера; amount - количество выставленных единиц валюты; left - не выкупленный остаток по ордеру относительно общего количества; taker_fee - комиссия взимается в тот момент, когда пользователь выкупает выставленное предложение другого пользователя; maker_fee - комиссия взимается в тот момент, когда пользователь выставляет новое предложение о покупке / продаже ордера; deal_stock - объем по сделке при покупке ордеров;

	deal_money - объем по сделке при продаже ордеров; deal_fee - комиссия по сделке.
Пример использования	<pre>query: `{pendingOrderDetail(pair: "BTCUSD", orderId: 980026){id, market, type, side, ctime, mtime, price, amount, left, taker_fee, maker_fee, deal_stock, deal_money, deal_fee}}`</pre>
Пример ответа	<pre>{ data: { pendingOrderDetail: { id: 980026, market: 'BTCUSD', type: 1, side: 1, ctime: 1561114877.058889, mtime: 1561122032.637261, price: 1325, amount: '1', left: '0.95', taker_fee: 0.002, maker_fee: 0.002, deal_stock: 0.05, deal_money: 66.25, deal_fee: 0.1325 } } }</pre>

Таблица 12 – Метод orderHistory (История ордеров пользователя)

Наименование метода	orderHistory
Описание метода	История ордеров пользователя
Тип протокола	HTTP
Тип метода	Приватный метод
Входящие параметры	pair - название пары валют; startTime - время начала суток; endTime - время окончания суток; offset - количество позиций, которые нужно пропустить от начала списка (по умолчанию 0); limit - количество отображаемых позиций (по умолчанию 50); side - статус ордера (лимитный ордер - 0, ордер на покупку - 1, ордер на продажу - 2).
Исходящие параметры	id - идентификатор ордера; ctime - время создания ордера; ftime - время закрытия ордера; market - название рынка; type - тип операции («1» / «sell» - продажа или «2» / «buy» - покупка); side - статус ордера (лимитный ордер - 0, ордер на покупку - 1, ордер на продажу - 2); price - цена за единицу валюты при покупке/продаже ордера; amount - количество выставленных единиц валюты; taker_fee - комиссия взимается в тот момент, когда пользователь выкупает выставленное предложение другого пользователя; maker_fee - комиссия взимается в тот момент, когда пользователь выставляет новое предложение о покупке / продаже ордера; deal_stock - объем по сделке при покупке ордеров; deal_money - объем по сделке при продаже ордеров; deal_fee - комиссия по сделке.

Пример использования	<pre>query: `{orderHistory(pair: "BTCUSD", startTime: 0, endTime: 0, offset: 0, limit: 50, side: "all") {id, ctime, ftime, market, type, side, price, amount, taker_fee, maker_fee, deal_stock, deal_money, deal_fee}}`</pre>
Пример ответа	<pre>{ data: { orderHistory: [{ id: 980024, ctime: 1561114863.52639, ftime: 1561118490.00507, market: 'BTCUSD', type: 1, side: 'asks', price: 1325, amount: 1, taker_fee: 0.002, maker_fee: 0.002, deal_stock: 1, deal_money: 1325, deal_fee: 2.65 }, { id: 980040, ctime: 1561119530.91628, ftime: 1561119530.91628, market: 'BTCUSD', type: 1, side: 'bids', price: 10800, amount: 0.005, taker_fee: 0.002, maker_fee: 0.002, deal_stock: 0.005, deal_money: 6.625, deal_fee: 0.00001 }] } }</pre>

Таблица 13 – Метод marketUpdated (Подписка на обновление рынков выбранной валюты)

Наименование метода	marketUpdated
Описание метода	Подписка на обновление рынков выбранной валюты
Тип протокола	Websocket
Тип метода	Публичный метод
Входящие параметры	currency - название валюты, входящей на рынок
Исходящие параметры	market - название рынка; stock - приобретаемый вид валюты; money - денежный эквивалент; open - цена по выбранной валютной паре на момент открытия Биржи; high - цена максимальная на дату проведения сделки по выбранной валютной паре;

	<p>low - цена минимальная на дату проведения сделки по выбранной валютной паре;</p> <p>lastPrice - цена по выбранной валютной паре на текущий момент;</p> <p>percentChange - изменение цены по выбранной валютной паре на текущий момент к цене по выбранной валютной паре на момент открытия Биржи за последние 24 часа в процентах;</p> <p>volume - объем проведенных торгов по видам валют.</p>
Пример использования	<pre>query: `subscription marketUpdated(\$currency: String!) { marketUpdated(currency: \$currency) { market stock money open high low lastPrice percentChange volume } }`, variables: { currency: 'BTC' },</pre>
Пример ответа	<pre>{ data: { marketUpdated: [{ market: 'BTCETH', stock: 'BTC', money: 'ETH', open: '10.5', high: '10.5', low: '10.5', lastPrice: '10.5', percentChange: '0.00', volume: '0' }] } }</pre>

Таблица 14 – Метод stateUpdated (Подписка на обновление информации по выбранному рынку)

Наименование метода	stateUpdated
Описание метода	Подписка на обновление информации по выбранному рынку
Тип протокола	Websocket
Тип метода	Публичный метод
Входящие параметры	market - название рынка
Исходящие параметры	<p>open - цена по выбранной валютной паре на момент открытия Биржи;</p> <p>market - название рынка;</p> <p>stock - приобретаемый вид валюты;</p> <p>money - денежный эквивалент;</p> <p>high - цена максимальная на дату проведения сделки по выбранной валютной паре;</p> <p>low - цена минимальная на дату проведения сделки по выбранной валютной паре;</p>

	lastPrice - цена по выбранной валютной паре на текущий момент; percentChange - изменение цены по выбранной валютной паре на текущий момент к цене по выбранной валютной паре на момент открытия Биржи за последние 24 часа в процентах; volume - объем проведенных торгов по видам валют.
Пример использования	<pre> query: `subscription stateUpdated(\$market: String!) { stateUpdated(market: \$market) { open market stock money high low lastPrice percentChange volume } }`, variables: { market: 'BTCUSD' }, </pre>
Пример ответа	<pre> { data: { stateUpdated: { open: '8480', market: 'BTCUSD', stock: null, money: null, high: '8485', low: '8480', lastPrice: '8483', percentChange: '0,35', volume: '20' } } } </pre>

Таблица 15 – Метод `chartUpdated` (Подписка на изменение графика сопоставления валютной пары по объему проданных/ приобретенных средств к цене проведенной сделки за указанный временной период в формате "японские свечи")

Наименование метода	chartUpdated
Описание метода	Подписка на изменение графика сопоставления валютной пары по объему проданных/ приобретенных средств к цене проведенной сделки за указанный временной период в формате "японские свечи"
Тип протокола	Websocket
Тип метода	Публичный метод
Входящие параметры	market - название рынка; interval - временной интервал котировок в секундах, отображаемых на графике цен: <ul style="list-style-type: none"> - 30 — 30 секунд; - 60 — 1 минута; - 120 — 2 минуты; - 300 — 5 минут; - 600 — 10 минут; - 900 — 15 минут; - 1800 — 30 минут;

	<ul style="list-style-type: none"> - 3600 — 1 час; - 14400 — 4 часа; - 86400 — 1 день; - 172800 — 2 дня.
Исходящие параметры	time - время совершения сделки за указанный временной период, отображаемое в формате "японские свечи" на графике; open - цена по выбранной валютной паре на момент открытия Биржи; high - цена максимальная на дату проведения сделки по выбранной валютной паре; low - цена минимальная на дату проведения сделки по выбранной валютной паре; close - цена по выбранной валютной паре на момент закрытия Биржи; volume - объем проведенных торгов по видам валют.
Пример использования	<pre>query: `subscription chartUpdated(\$market: String!, \$interval: Int!) { chartUpdated(market: \$market, interval: \$interval) { time open high low close volume } }`, variables: { market: 'BTCUSD', interval: 60 },</pre>
Пример ответа	<pre>{ data: { chartUpdated: [{ time: 1573834500, open: 8480, high: 8485, low: 8480, close: 8483, volume: 0,35 }] } }</pre>

Таблица 16 – Метод orderBookUpdated (Подписка на обновление списка ордеров по продаже/ покупке)

Наименование метода	orderBookUpdated
Описание метода	Подписка на обновление списка ордеров по продаже/ покупке
Тип протокола	Websocket
Тип метода	Публичный метод
Входящие параметры	market - название рынка; limit - количество отображаемых позиций (по умолчанию 50, не должно превышать 1000); precision - шаг; объединение близких по цене предложений в одну позицию.
Исходящие параметры	asks - список ордеров на продажу (предложение); bids - список ордеров на покупку (спрос); price - цена за единицу валюты при покупке/продаже ордера; volume – объем; количество выставленных единиц; entirety – признак целостности книги заявок: <ul style="list-style-type: none"> - true - получение полного списка заявок (при первом выполнении запроса); - false - получение обновлений к списку заявок.
Пример использования	<pre>query: `subscription</pre>

	<pre> orderBookUpdated(\$market: String!, \$limit: Int! = 50, \$precision: String! = "0") { orderBookUpdated(market: \$market, limit: \$limit, precision: \$precision) { asks {price volume} bids {price volume} entirety } }`, variables: { market: 'BTCUSD', limit: 50, precision: '0' }}, </pre>
Пример ответа	<pre> { data: { orderBookUpdated: { asks: [{ price: '8483', volume: '2' }, { price: '8482', volume: '3.5' }], bids: [{ price: '8481', volume: '2' }], entirety: true } } } </pre>

Таблица 17 – Метод dealsUpdated (Подписка на обновление списка проведенных сделок по продаже/ покупке)

Наименование метода	dealsUpdated
Описание метода	Подписка на обновление списка проведенных сделок по продаже/ покупке
Тип протокола	Websocket
Тип метода	Публичный метод
Входящие параметры	market - название рынка
Исходящие параметры	id - идентификатор сделки; time - время заключения сделки; price - цена за единицу валюты при покупке/продаже ордера; amount - количество выставленных единиц валюты; type - тип операции («1» / «sell» - продажа или «2» / «buy» - покупка).
Пример использования	<pre> query: `subscription dealsUpdated(\$market: String!) { dealsUpdated(market: \$market) { id time price amount type } }`, variables: { market: 'BTCUSD' }, </pre>
Пример ответа	<pre> { data: { dealsUpdated: [{ id: '200087', time: '1573461119.464421', price: '8483', </pre>

	<pre> amount: '10', type: 'buy' }, { id: '200086', time: '1573461119.464212', price: '8470', amount: '25', type: 'sell' }] } }</pre>
--	---

Таблица 18 – Метод balanceUpdated (Подписка на обновление баланса пользователя)

Наименование метода	balanceUpdated
Описание метода	Подписка на обновление баланса пользователя
Тип протокола	Websocket
Тип метода	Приватный метод
Входящие параметры	currencyList - Список валют, по которым необходимо отслеживать изменения
Исходящие параметры	currency - название валюты, входящей на рынок; available - доступный баланс пользователя; frozen - зарезервированный баланс по выставленным ордерам.
Пример использования	<pre> query: `subscription balanceUpdated(\$currencyList: [String!]!) { balanceUpdated(currencyList: \$currencyList) { currency available frozen } }`, variables: { currencyList: ['BTC'] },</pre>
Пример ответа	<pre> { data: { balanceUpdated: [{ currency: 'BTC', available: '9959.83141406', frozen: '55.99893348' }] } }</pre>

Таблица 19 – Метод orderUpdated (Подписка на изменение статуса активных ордеров пользователя)

Наименование метода	orderUpdated
Описание метода	Подписка на изменение статуса активных ордеров пользователя
Тип протокола	Websocket
Тип метода	Приватный метод
Входящие параметры	markets - Список рынков, ордера по которым требуется отслеживать
Исходящие параметры	id - идентификатор ордера; status - статус выполнения ордера («pending» - в ожидании выполнения, «updated» - частично выполнен, «finished» - полностью выполнен, «canceled» - отменен); market - название рынка;

	<p> ctime - время создания ордера; mtime - время последнего изменения по ордеру; ftime - время закрытия ордера; type - тип ордера («1» / «лимитный» или «2» / «рыночный»); side - тип операций по ордерам («asks» - продажа, «bids» - покупка); price - цена за единицу валюты при покупке/продаже ордера; amount - количество выставленных единиц валюты; taker_fee - комиссия взимается в тот момент, когда пользователь выкупает выставленное предложение другого пользователя; maker_fee - комиссия взимается в тот момент, когда пользователь выставляет новое предложение о покупке / продаже ордера; deal_fee - комиссия по сделке; deal_stock - объем по сделке при покупке ордеров; deal_money - объем по сделке при продаже ордеров; left - не выкупленный остаток по ордеру относительно общего количества. </p> <p> <u>Для выбора типа ордера «стоп-заявка»</u> по выставляемым ордерам продажи/ покупки проставить галочку в поле «Stop/Take profit», указать тип сделки «ask/bid» и указать максимально допустимое значение в поле «Стоп-цена». </p> <p> triggerType - флаг выбора типа ордера «стоп-заявка» (0 - флаг не установлен, 14 - «стоп-ордер» типа «ask», 15 - «стоп-ордер» типа «bid»); triggerPrice – максимально допустимое значение в поле «Стоп-цена». </p>
Пример использования	<pre> 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: ['BTCUSD'] }, </pre>
Пример ответа	<pre> { data: { orderUpdated: { id: 3874821, status: 'pending', market: 'BTCUSD', ctime: '1574073768.52127', mtime: '1574073768.52127', ftime: null, </pre>

```

        type: 1,
        side: 'bids',
        price: '1',
        amount: '6',
        taker_fee: '0.001',
        maker_fee: '0.001',
        deal_fee: '0',
        deal_stock: '0',
        deal_money: '0',
        left: '6',
        triggerType: 0,
        triggerPrice: '' } } }
{ data:
  { orderUpdated:
    { id: 3874840,
      status: 'finished',
      market: 'BTCUSD',
      ctime: '1574075948.929811',
      mtime: '1574075948.929811',
      ftime: null,
      type: 1,
      side: 'asks',
      price: '2.5',
      amount: '5',
      taker_fee: '0.001',
      maker_fee: '0.001',
      deal_fee: '0',
      deal_stock: '0',
      deal_money: '0',
      left: '5',
      triggerType: 14,
      triggerPrice: '2.4' } } }
{ data:
  { orderUpdated:
    { id: 3874836,
      status: 'finished',
      market: 'BTCUSD',
      ctime: '1574075392.222283',
      mtime: '1574075392.222283',
      ftime: null,
      type: 1,
      side: 'bids',
      price: '2.5',
      amount: '1',
      taker_fee: '0.001',
      maker_fee: '0.001',
      deal_fee: '0',
      deal_stock: '0',
      deal_money: '0',
      left: '1',

```

```
triggerType: 15,  
triggerPrice: '2.45' } } }
```

3.3 Выходные данные

В разделе «Структура запроса» приведены несколько вариантов вывода данных (подробнее см. раздел 3.1, абзац «Вывод информации в веб-консоль»), в данном разделе показаны полученные результаты на примере выполнения одного из методов.

Формат вывода ответа приходит в виде сообщения в веб - консоль с параметрами согласно формату запроса (возможные типы вывода информации в console):

1) Вывод перечня параметров в виде массива и создание нового объекта, содержащего данные массива.

```
const {data:{tradeHistory:massivArguments}}=body  
console.log(massivArguments)  
console.log('createObjects',{massivArguments})
```

Формат ответа:

```
[ { amount: '0.005',  
  price: '1325',  
  time: '1561379389.82101',  
  type: 'buy' },  
  { amount: '0.005',  
    price: '1325',  
    time: '1561118474.63568',  
    type: 'buy' } ]  
createObjects { massivArguments:  
  [ { amount: '0.005',  
    price: '1325',  
    time: '1561379389.82101',  
    type: 'buy' },  
    { amount: '0.005',  
      price: '1325',  
      time: '1561118474.63568',  
      type: 'buy' } ] ] }
```

2) Вывод полного списка параметров запроса с присвоенными значениями.

```
console.log(body.data)
```

Формат ответа:

```
{ tradeHistory:  
  [ { amount: '0.005',  
    price: '1325',  
    time: '1561379389.82101',  
    type: 'buy' },  
    { amount: '0.005',  
      price: '1325',  
      time: '1561118474.63568',  
      type: 'buy' } ] ] }
```

3) Вывод структуры переданного объекта с раскрытием данных запроса до указанного уровня глубины.

```
console.dir(response.body,{depth:3})
```

Формат ответа:

```
{ data:
  { tradeHistory:
    [ { amount: '0.005',
        price: '1325',
        time: '1561379389.82101',
        type: 'buy' },
      { amount: '0.005',
        price: '1325',
        time: '1561118474.63568',
        type: 'buy' } ] } }
```

ПРИМЕЧАНИЕ:

Для корректного отображения полученных значений необходимо указать максимальный уровень глубины. Пример неверного указания уровня вложенности:

```
{ data: { tradeHistory: [Array] } }
```

4) Передача настроек сервера при выполнении запроса.

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

Развёрнутый ответ о настройках пользователя. В приведенном примере показан небольшой фрагмент, т.к. полный текст ответа включает несколько страниц.

```
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:
```

5) Получение сообщения об ошибке. В случае успешного выполнения запроса выводится ответ "null".

```
console.log(error)
```

Ответ содержит текст ошибки или информацию об успешном выполнении запроса – «null».

```
null
```

6) Получение списка переданных в запросе параметров и текста запроса.

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

Ответ содержит настройки, переданные серверу (адрес url, формат передачи времени в Unix в строке «nonce», публичный apiKey и подпись).

```
{ contentType: 'application/json',  
  url: 'http:// exs.cash /api/v1/',  
  headers:  
    { nonce: 1561381313,  
      apiKey:  
        'Выведен публичный ключ Token',  
      signature:  
        ' Выведен приватный ключ Token' },  
  body:  
    { query:  
      '{tradeHistory(pair: "BTCUSD", limit: 10, offset: 100){amount, price, time,  
type}}' },  
    json: true }
```

7) Информационное сообщение о том, что подписка на метод произведена, лог в состоянии ожидания событий.

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

Формат ответа:

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

8) Вывод полного списка параметров запроса с присвоенными значениями и добавлением глубины раскрытия уровней; информация получена с использованием библиотеки utils от Node.

```
console.log('Received event: ', util.inspect(eventData, { depth: 10 }))
```

Формат ответа:

```
Received event: { data:  
  { orderBookUpdated:  
    { asks:  
      [ { price: 8483, volume: '2' }, { price: '8482, volume: '3.5' } ],  
      bids: [ { price: '8481, volume: '2' } ],  
      entirety: true } } }
```

9) Вывод полного списка параметров запроса с присвоенными значениями и добавлением глубины раскрытия уровней; информация получена с

использованием стандартной функцией Java-скрипт. Формат вывода предыдущего и текущего лога аналогичен.

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

Формат ответа:

```
{ data:
  { orderBookUpdated:
    { asks:
      [ { price: '8483', volume: '2' },
        { price: '8482', volume: '3.5' } ],
      bids: [ { price: '8481', volume: '2' } ],
      entirety: true } } }
```

Приложение А. Тест HTTP-запроса для API Биржи

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

const apiKey = 'Ввести публичный ключ Token'
const apiSecret = 'Ввести приватный ключ Token'
const nonce = (Date.now() / 1000) | 0 // В секундах
const body = {
  query: `{currencyList{name, prec}}`, //Получение списка валют, публичный
  // query: `{tradeHistory(pair: "BTCUSD", limit: 10, offset: 100){amount, price, time,
  type}}`, //Получение списка валют, публичный
  // query: `{marketList{name, stock, money, fee_prec, stock_prec, money_prec,
  min_amount}}`, //Получение списка маркетов с их параметрами, публичный
  // query: `{orderBook(pair: "BTCUSD", limit: 50, precision: "0"){asks{price, volume},
  bids{price, volume}}}`, //Получение стакана, публичный
  // query: `mutation{putLimitOrder(market: "BTCUSD", side: "bids", amount: "0.005",
  price: "8800") {status, id, deal_stock, deal_money, deal_fee}}`, //Выставление ордера,
  приватный
  // query: `mutation{cancelOrder(pair: "BTCUSD", orderId: 980023){id, ctime, mtime,
  market, type, side, price, amount, taker_fee, maker_fee, deal_stock, deal_money,
  deal_fee}}`, //Отменить ордер, приватный
  // query: `{balance{currency, available, frozen}}`, //Баланс пользователя, приватный
  // query: `{pendingOrders(pair: "BTCUSD", offset: 0, limit: 100){id, market, mtime,
  ctime, deal_fee, deal_stock, deal_money, taker_fee, maker_fee type, side, price, amount,
  left}}`, //Открытые ордера пользователя, приватный
  // query: `{pendingOrderDetail(pair: "BTCUSD", orderId: 961367){id, market, type, side,
  ctime, mtime, price, amount, left, taker_fee, maker_fee, deal_stock, deal_money,
  deal_fee}}`, //Открытый ордер пользователя, приватный
  // query: `{orderHistory(pair: "BTCUSD", startTime: 0, endTime: 0, offset: 0, limit: 50,
  side: "all") {id, ctime, ftime, market, type, side, price, amount, taker_fee, maker_fee,
  deal_stock, deal_money, deal_fee}}`, //История ордеров пользователя, приватный

  //query: `{marketList{name, stock, money, fee_prec, stock_prec, money_prec,
  min_amount}
  //orderBook(pair: "BTCUSD", limit: 50, precision: "0"){asks{price, volume}, bids{price,
  volume}}}`, //Объединенный запрос
}

const gurl = 'https://exs.cash/api/v1/'

const signature = `${gurl}${nonce}${JSON.stringify(body)}`

const shex = crypto
  .createHmac('sha512', apiSecret)
  .update(signature)
  .digest('hex')

const options = {
  contentType: 'application/json',
  url: 'https://exs.cash/api/v1/',
```



```

headers: {
  nonce,
  apiKey,
  signature: shex,
},
body,
json: true,
}

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

  // const {data:{tradeHistory:massivArguments}}=body
  // console.log(massivArguments)
  // console.log('createObjects',{massivArguments})

  console.log(body.data)
  // console.dir(response.body,{depth:10})
  // console.log(response)
  // console.log(error)
  // console.log(options)
})

```

Приложение Б. Тест Websocket -запроса для API Биржи

```
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 util = require("util")
```

```
const apiKey = 'Ввести публичный ключ Token'
const secretKey = 'Ввести приватный ключ Token'
```

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

```
const requestTable = {
  marketUpdated: {
    query: `
      subscription marketUpdated($currency: String!) {
        marketUpdated(currency: $currency) {
          market
          stock
          money
          open
          high
          low
          lastPrice
          percentChange
          volume
        }
      }
    `,
    variables: { currency: 'BTC' },
  },
  stateUpdated: {
    query: `
      subscription stateUpdated($market: String!) {
        stateUpdated(market: $market) {
          open
          market
          stock
          money
          high
          low
          lastPrice
          percentChange
          volume
        }
      }
    `,
    variables: { market: 'BTCUSD' },
  },
},
```

```

chartUpdated: {
  query: `
    subscription chartUpdated($market: String!, $interval: Int!) {
      chartUpdated(market: $market, interval: $interval) {
        time
        open
        high
        low
        close
        volume
      }
    }
  `,
  variables: { market: 'BTCUSD', interval: 60 },
},
orderBookUpdated: {
  query: `
    subscription orderBookUpdated($market: String!, $limit: Int! = 50,
    $precision: String! = "0") {
      orderBookUpdated(market: $market, limit: $limit, precision:
      $precision) {
        asks {
          price
          volume
        }
        bids {
          price
          volume
        }
        entirety
      }
    }
  `,
  variables: { market: 'BTCUSD', limit: 50, precision: '0' },
},
dealsUpdated: {
  query: `
    subscription dealsUpdated($market: String!) {
      dealsUpdated(market: $market) {
        id
        time
        price
        amount
        type
      }
    }
  `,
  variables: { market: 'BTCUSD' },
},
balanceUpdated: {
  query: `
    subscription balanceUpdated($currencyList: [String!]!) {
      balanceUpdated(currencyList: $currencyList) {
        currency
        available
      }
    }
  `

```

```

        frozen
      },
    },
    variables: { currencyList: ['BTC'] },
  },
  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: ['BTCUSD'] },
  },
}

```

```

const wsUrl = 'ws://exs.cash/api/v1/'
const httpUrl = 'http:// exs.cash/api/v1/'

```

```

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

  const serverSignature = crypto
    .createHmac('sha512', secretKey)
    .update(data)
    .digest('hex')

  return serverSignature
}

```

```

const request = requestTable.orderBookUpdated

```

```

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

```

```

        secretKey,
    })

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

    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
      )

      subscriptionObservable.subscribe(eventData => {

        console.log(`Subscribed successfully, waiting for messages...`)
        console.dir(eventData, {depth:10})
        //console.log('Received event: ', util.inspect(eventData, { depth: 10 }))
      })
    }

    subscribe()

```

СПИСОК СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

API (Application Programming Interface)	– набор публичных свойств и методов для взаимодействия с другими объектами в приложении.
HTTP (HyperText Transfer Protocol)	– протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

СПИСОК ТЕРМИНОВ И ОПРЕДЕЛЕНИЙ

API-keys (application programming interface keys) – совокупность публичного и секретного ключей Token, создание новых цепочек блоков происходит на основании подтверждения операций секретным ключом.

ICO (Initial coin offering) – форма привлечения инвестиций в виде продажи инвесторам фиксированного количества новых единиц криптовалют, полученных разовой или ускоренной эмиссией.

Token – создание бота, имеющего доступ к паре публичного и секретного ключей. Бот обеспечивает выполнение заданных типов операций на бирже от имени пользователя.

Websocket - протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени. В отличие от протокола HTTP данные по протоколу Websocket приходят, когда происходит событие, на которое производилась подписка. До закрытия соединения подписка остается активной и ожидает ответы сервера.

Блокчейн (block chain) – децентрализованная база данных, в которой устройства хранения данных не подключены к общему серверу. Пользователи могут редактировать только те части цепочки блоков, к которым они имеют доступ на основании предоставления ключей. Шифрование обеспечивает синхронизацию копий распределённой цепочки блоков у всех пользователей.

Криптовалюта – внутренняя расчётная единица валюты, которая добывается с помощью майнинга и привязана к одному/нескольким кошелькам. При выполнении любых операций с криптовалютой производится запись операций в блокчейн.

Лимитный ордер – тип ордера, который указывает брокеру максимальную цену, по которой пользователь готов купить или минимальную цену, по которой готов продать валюту.

Майнинг (mining) – серии вычислений с перебором параметров для нахождения «хеша» с заданными свойствами, которые необходимы для обеспечения функционирования криптовалютных платформ.

Мейкер (maker) – часть трейдеров, действующих на торговых площадках и предоставляющих для актива дополнительную ликвидность, способствуя смещению котировок в сторону повышения или понижения. Трейдеры выставляют отложенные ордера, ценовой уровень которых выше/ниже существующих значений. Биржа поощряет данную категорию трейдеров отсутствием комиссий по сделкам или снижением процентов до минимума.

Тейкер (taker) – часть трейдеров, действующих на торговых площадках, снижающих ликвидность актива из-за уменьшения количества выставленных предложений. Трейдеры совершающих торговые операции незамедлительно при наличии отложенного ордера с соответствующими условиями по рыночной цене.

Форжинг (Forging) / Минтинг(Minting) – создание в криптовалютах новых блоков на основании подтверждения доли владения с возможностью получить вознаграждение в форме новых единиц и комиссионных сборов.

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, позволяет хранить пары (ключ, значение) и выполнять три вида операций: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу. Каждый хеш уникален, не связан по значению с предыдущим.

Эмиссия – выпуск в обращение новых денег.