# API development



## Задание

Нужно разработать класс для работы с API биржи в среде Node.js. Данные должны приходить в реальном времени через протокол WebSocket. Остальные действия с биржой (выставление ордера, получение балансов и т.д) могут происходить как по вэбсокету, так и через REST API.

## Требования к коду/разработке

- Версия Node.js >= v8.16.0
- Использовать ES6+ спецификацию
- Классы описываются с заглавной буквы
- Методы описываются с маленькой буквы
- Классы и методы должны комментироваться
- Комментарии должны основываться на синтаксисе jsDoc
- Использовать «строгий режим» ('use strict')
- Предпочтительно использование строгого сравнения
- Простота и чистота кода

## Используемые библиотеки

Для разработки использовать следующие библиотеки:

- Работа с вэбсокетами - WS WebSocket library (https://www.npmjs.com/package/ws)
- Работа с HTTP запросами - Request library (https://www.npmjs.com/package/request)
- Математические операции - BigNumber.js (https://www.npmjs.com/package/bignumber.js)
- Работа с датой - Moment (https://www.npmjs.com/package/moment)
- Уникальный идентификатор - Uuid (https://www.npmjs.com/package/uuid)
- Валидация данных - Ajv (https://www.npmjs.com/package/ajv)

## Разработка

Все классы для работы с API бирж должны иметь названия вида ExchangeAPI (где Exchange - название биржи с большой буквы) и должны наследоваться от базового класса API.

Базовая структура класса ExchangeAPI состоит из конструктора, методов onOpen, onClose, onMessage и методов, каждый из которых отвечает за определенные действия на бирже (авторизация, выставление ордеров, получение баланса и т.д). Полученные данные по ордербукам сохраняются в локальный ордербук.

Технические требования:

- Timestamp должен быть в милисекундах
- Для генерации id нужно использовать библиотеку UUID, а именно Version 1 (timestamp), если это возможно
- Использовать синтаксис async/await для работы с промисами
- Соблюдать порядок объявления методов (если в методе используется другой метод этого класса, то используемый метод должен быть объявлен ниже текущего метода)
- Дополнительные объявленные методы должны начинаться с _ (нижнее подчёркивание)
- Комментирование - все методы, которые нужно реализовать, уже хорошо закомментированы в классе API и не требуются в комментариях, но дополнительные объявленные методы должны быть закомментированы jsDoc'ом.

### Конструктор и методы onOpen, onClose, onMessage

Метод onOpen вызывается системой тогда, когда происходит открытие вэбсокетного соединения с биржой.

Метод onClose вызывается тогда, когда происходит закрытие вэбсокетного соединения с биржой. Иногда при активном вэбсокетном соединении происходит закрытие соединения по вине биржи. В этом случае система производить автоматическое переподключение, но перед этим после закрытия соединения, система вызывает метод onClose, в котором должна быть прописана логика очистки данного

класса (освобождение объектов и т.д). Поэтому всю логику очищения стоит прописывать именно в этом методе.

Метод onMessage вызывается тогда, когда приходит сообщение от биржи по вэбсокетному соединению. Аргумент message в методе on Message приходит в виде преобразованной строки JSON. Прежде чем работать с аргументом message, нужно его распарсировать.

Семантика методов:

```
/**
 * Constructor
 * @constructs ExchangeAPI
 * @param {string} name - exchange name
 * @param {string} apiKey - API key
 * @param {string} apiSecret - secret key
 */
constructor(name, apiKey, apiSecret) {
  super(name, apiKey, apiSecret);
}

/**
 * @typedef {Object} WSObject
 * @property {number} id - unique number of WebSocket connection
 * @property {string} wsUrl - WebSocket url
 * @property {ws} ws - WebSocket object
 * @property {string} restUrl - REST API url
 * @property {request} rest - Request object
 * @property {Object} reconnectionTimer - object returned by method
setTimeout
 * @property {boolean} error - true if error occures in WebSocket
connection
 * @property {boolean} isApiExplicitlyClosed - true if WebSocket
connection was closed explicitly
 */

/**
  * Called on WebSocket open event
  * @param {WSObject} wsData - WebSocket connection data
  */
onOpen(wsData) {}

/**
  * Called on WebSocket close event
  * @param {WSObject} wsData - WebSocket connection data
  */
onClose(wsData) {}

/**
  * Called when WebSocket message event occurred
  * @param {WSObject} wsData - WebSocket connection data
  * @param message - message received from exchange
  */
onMessage(wsData, message) {}
```

- WSObject.ws - вэбсокетное соединение установленное WS библиотекой (пример: WSObject.ws = new WebSocket(url))
- WSObject.rest - объект для работы с REST API (request библиотека)

Получение данных

Все данные, которые приходят с биржи, посылаются в метод onMessage и в этом методе должна происходить обработка этих данных.

Данные, которые получили по подписке на ордербук биржи, нужно сохранять/удалять в локальном ордербуке. Для добавления/изменения данных, используется метод updateOrderBook базового класса API. Для удаления данных, используется метод deleteOrderBook.

Семантика метода updateOrderBook:

```
/**
  * Add order or modify existing order
  * @param {string} exchangeSymbol - exchange symbol
  * @param {string} type - "bid" or "ask"
  * @param {number} price - order price
  * @param {amount} amount - amount
  */
updateOrderBook(exchangeSymbol, type, price, amount)
```

При вызове метода updateOrderBook, система добавит новый ордер, если данного ордера нет в локальном ордербуке. Если же ордер уже имеется в локальном ордербуке, тогда система этот ордер изменит. Ордера в ордербуке идентифицируются по цене.

Семантика метода deleteOrderBook:

```
/**
  * Delete order
  * @param {string} exchangeSymbol - exchange symbol
  * @param {string} type - "bid" or "ask"
  * @param {number} price - order price
  */
deleteOrderBook(exchangeSymbol, type, price)
```

При вызове метода deleteOrderBook, система удалит запись из локального ордербука. Поиск на удаление ведется по цене.

Если возникает ошибка в методах onOpen, onClose, onMessage, рекомендуется вызывать метод error базового класса API.

Семантика метода error:

```
/**
  * Must be called when API gets an error
  * @param {string} message - (optional) message
  * @param {Object} error - error object
  */
error(error, message = '')
```

В случае, если нужно очистить локальный ордербук от определенного символа (к примеру произошла отписка от данного символа), необходимо вызвать метод clearOrderBook.

Семантика метода clearOrderBook:

```
/**
  * Clear orderbook by symbol
  * @param {string} symbol - exchange symbol
  */
clearOrderBook(symbol)
```

Разработка методов

Ниже приведены методы, которые нужно разработать для реализации функционала API биржи:

- getExchangeStatus - получение статуса биржи

```
/**
 * @typedef {Object} ErrorObject
 * @property {object} error - error
 * @property {string} message - error message
 */

/**
  * @typedef {Object} StatusObject
  * @property {boolean} status - exchange status
  */
/**
  * Get exchange status
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @return {Promise.<StatusObject | ErrorObject>}
  */
async getExchangeStatus(wsData) {}
```

Не на всех биржах есть метод в API, который проверяет статус биржи. Поэтому статус биржи (работает биржа или нет) можно проверять каким-нибудь “легким” запросом.

- ping - отправка ping на сервер биржи по WebSocket соединению

```
/**
  * @typedef {Object} PongObject
  * @property {string} result - 'pong' message
  */
/**
  * Make ping request to WebSocket connection
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @return {Promise.<PongObject | ErrorObject>}
  */
async ping(wsData) {}
```

Если данного функционала нет на бирже, тогда этот метод не имплементировать.

- authorize - авторизация на бирже

```
/**
  * Authorization
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @return {Promise.<StatusObject | ErrorObject>}
  */
async authorize(wsData) {}
```

Если авторизации на бирже нет (все запросы к API подписываются), тогда этот метод можно не реализовывать.

- subscribeOrderbooks - подписка на ордербуки

```
/**
  * @typedef {Object} SubscriptionObject
  * @property {Array<string>} symbols - array of symbols
  */
/**
  * Subscribe to order books
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @param {Array<string>} symbols - array of symbols
  * @return {Promise.<SubscriptionObject | ErrorObject>}
  */
async subscribeOrderbooks(wsData, symbols) {}
```

Пример подписки: subscribeOrderbooks(wsData, ['ETHBTC', 'BTCUSDT']);

- makeOrder - выставление ордера

```
/**
  * @typedef {Object} OrderObject
  * @property {string} orderId - exchange orderId
  */
/**
  * Make order
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @param {string} symbol - cryptocurrency pair, example: "ETHBTC"
  * @param {number} amount - cryptocurrency amount
  * @param {number} bid - bid price
  * @param {number} ask - ask price
  * @return {Promise.<OrderObject | ErrorObject>}
  */
async makeOrder(wsData, symbol, amount, bid, ask) {}
```

Здесь идёт речь о моментальном выставлении ордера (market order). Если amount > 0, тогда должен выставиться ордер на покупку, если amount < 0, тогда ордер на продажу. При выставлении market ордера, параметры bid и ask обычно не требуются.

- getDepositInfo - получение адреса кошелька определенной валюты

```
/**
  * @typedef {Object} DepositObject
  * @property {string} currency - currency, example: "BTC", "ETH"
  * @property {string} address - wallet address
  * @property {string | null} payment_id - If set, it is required for
deposit
  */
/**
  * Get information about account
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @param {string} currency - currency, example: "BTC", "ETH"
  * @return {Promise.<DepositObject | ErrorObject>}
  */
async getDepositInfo(wsData, currency) {}
```

Некоторые валюты, такие как XRP, имеют основной адрес и дополнительный. В документации API бирж дополнительный адрес может называться как memo, payment_id или по другому. Поэтому, если в валюте имеется дополнительный адрес, он должен возвращаться в атрибуте payment_id объекта DepositObject. Если его нет, то payment_id=null.

- makeWithdraw - перевод валюты на определенный кошелек

```
/**
  * @typedef {Object} TransactionObject
  * @property {string | null} id - transaction id (null if exchange does
not provide any transaction id)
  */
/**
  * Withdraw
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @param {DepositObject} depositInfo - information about account where
currency is sending
  * @param {number} amount - cryptocurrency volume
  * @return {Promise.<TransactionObject | ErrorObject>}
  */
async makeWithdraw(wsData, depositInfo, amount) {}
```

Как было описано выше (см. метод getDepositInfo), некоторые валюты, такие как XRP, требуют кроме основного адреса, дополнительный. Поэтому объект depositInfo должен содержать дополнительный адрес в атрибуте payment_id.

- getWithdrawHistory - получение истории переводов

```
/**
  * @typedef {Object} HistoryObject
  * @property {string} id - transaction unique identifier as assigned by
exchange
  * @property {string} currency - currency, example: "BTC", "ETH"
  * @property {number} amount - amount
  * @property {number} status - transaction status (1 is "pending", 2 is
"failed", 3 is "completed")
  * @property {string | null} address - destination address (null if no
address provided - can be on deposit history)
  * @property {string | null} payment_id - additional address (null if
no additional address provided)
  * @property {string} hash - blockchain transaction hash (same as TXID
or transaction id)
  */
/**
  * Get withdraw history
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @param {string | null} currency - currency, example: "BTC", "ETH"
  * @param {string | null} id - transaction id assigned by exchange
  * @param {number} limit - number of deposit records
  * @return {Promise.<Array<HistoryObject>>}
  */
async getWithdrawHistory(wsData, currency = null, id = null, limit =
constant.HISTORY_LIMIT) {}
```

- getDepositHistory - получение истории пополнения

```
/**
  * @typedef {Object} HistoryObject
  * @property {string} id - transaction unique identifier as assigned by
exchange
  * @property {string} currency - currency, example: "BTC", "ETH"
  * @property {number} amount - amount
  * @property {number} status - transaction status (1 is "pending", 2 is
"failed", 3 is "done", , 4 is "unknown")
  * @property {string} address - destination address
  * @property {string} hash - blockchain transaction hash (same as TXID
or transaction id)
  */
/**
  * Get deposit history
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @param {string | null} currency - currency, example: "BTC", "ETH"
  * @param {string | null} id - transaction id assigned by exchange
  * @param {number} limit - number of deposit records
  * @return {Promise.<Array<HistoryObject>>}
  */
async getDepositHistory(wsData, currency = null, id = null, limit =
constant.HISTORY_LIMIT) {}
```

- getBalance - получение баланса

```
/**
  * Get exchange balance (indexes are currencies, values are amounts)
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @return {Promise.<Array<string, Number>>}
  */
async getBalance(wsData) {}
```

- makeInternalTransfer - перевод валюты внутри биржи между счетами (main, trading, …)

```
/**
  * Make internal transfer between internal accounts
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @param {string} currency - cryptocurrency
  * @param {number} amount - amount of transfer
  * @param {number} type - type of transfer (see transferType in
constants)
  * @return {Promise.<TransactionObject>}
  */
async makeInternalTransfer(wsData, currency, amount, type) {}
```

Функционал перевода валюты внутри биржи может быть не во всех биржах. Поэтому метод makeInternalTransfer имплементируется только в том случае, если у биржи имеется данный функционал. На некоторых биржах может быть несколько аккаунтов у каждой валюты, к примеру main аккаунт и trading аккаунт. На main аккаунте происходит пополнение, снятие денег, а на trading аккаунте происходит только торговля.

- getTradeHistory - получение данных по выполненным ордерам

```
/**
  * @typedef {Object} TradeObject
  * @property {string} id - trade id
  * @property {string} orderId - order id
  * @property {string} symbol - exchange symbol
  * @property {number} price - price
  * @property {number} amount - amount
  * @property {number} timestamp - trade timestamp
  * @property {string} type - 'buy' or 'sell'
  * @property {number | null} fee - trade fee
  */
/**
  * Get all trades of specific order ID and symbol
  * @async
  * @param {WSObject} wsData - WebSocket connection data
  * @param {string} symbol - exchange symbol (if not null, filter by
symbol)
  * @param {string | null} orderId - order ID (if not null, filter by
order ID)
  * @param {number} limit - number of trade records
  * @return {Promise.<Array<TradeObject>>}
  */
async getTradeHistory(wsData, symbol, orderId = null, limit =
constant.HISTORY_LIMIT) {}
```

Каждый ордер на бирже может быть выполнен одним или несколькими трейдами. Метод getTradeHistory возвращает именно трейды, а не ордера.

При реализации проекта, используйте типы ошибок и константы, которые уже приведены в проекте. Если требуется описать новые константы или ошибки, специфичные для определенной биржи, описывать их нужно в отдельных файлах.

> Если один из вышеперечисленных методов нельзя определить как описано выше, тогда нужно об этом сообщить для уточнения аннотации этих методов.
>
> Если имеются какие-то неточности в данном задании, также нужно об этом сообщить.

## Каркас API

К данному ТЗ прилагаю проект, который реализует упрощенную структуру работы API.

Описание проекта:

- Core/API.js - базовый класс API с описанием всех методов
- Core/constants.js - общие константы (константы, которые могут быть использованы при реализации API любой биржи)
- Core/errors.js - типы ошибок, которые могут быть переданы в конструктор класса Error
- Core/core.js - логика
- client.js - стартовый скрипт, в котором приведены примеры вызовов методов API
- config.js - конфигурационный файл, содержащий API ключи биржи

Проект можно найти в репозитории Bitbucket: https://bitbucket.org/redaseo/api-development