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.

Криптовалютная биржа

Биржа: Gate.io
Сайт: https://www.gate.io/

Требования к коду/разработке

- 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')
- 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)

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.

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, мы используем 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 при получении данных, 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 = '')
```

Метод error рекомендуется вызывать всегда, когда возникают ошибки. Это нужно для того, чтобы все ошибки логировались и имели одинаковую структуру.

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
  * @return {Promise.<StatusObject | ErrorObject>}
  */
getExchangeStatus() {}
```

- authorize - авторизация на бирже

```
/**
  * Authorization
  * @override
  * @param {object} ws - WebSocket object
  * @return {Promise.<undefined | ErrorObject>}
  */
authorize(ws) {}
```

- subscribeOrderbook - подписка на ордербук

```
/**
 * @typedef {Object} SubscriptionObject
 * @property {string} symbol - exchange symbol
 */

/**
  * Subscribe to orderbook
  * @override
  * @param {object} ws - WebSocket object
  * @param {string} symbol - cryptocurrency pair, example: "ETHBTC"
  * @return {Promise.<SubscriptionObject | ErrorObject>}
  */
subscribeOrderbook(ws, symbol) {}
```

- makeOrder - выставление ордера

```
/**
  * Make order
  * @override
  * @param {object} ws - WebSocket object
  * @param {string} symbol - cryptocurrency pair, example: "ETHBTC"
  * @param {number} amount - cryptocurrency volume
  * @param {number} bid - bid price
  * @param {number} ask - ask price
  * @return {Promise.<undefined | ErrorObject>}
  */
makeOrder(ws, symbol, amount, bid, ask) {}
```

- 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
  * @param {string} currency - currency, example: "BTC", "ETH"
  * @return {Promise.<DepositObject | ErrorObject>}
  */
getDepositInfo(currency) {}
```

- makeWithdraw - перевод валюты на определенный кошелек

```
/**
 * @typedef {Object} TransactionObject
 * @property {string} id - transaction id
 */

/**
  * Withdraw
  * @param {object} depositInfo - information about account
where currency is sending
  * @param {number} amount - cryptocurrency volume
  * @param {string} currency - currency, example: "BTC", "ETH"
  * @return {Promise.<TransactionObject | ErrorObject>}
  */
makeWithdraw(depositInfo, amount, currency) {}
```

- 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} address - destination address
 * @property {string} hash - transaction hash
 */

/**
  * Get withdraw history
  * @param {string} currency - currency, example: "BTC", "ETH"
  * @param id - if specified, method returns data of specific withdraw
transaction
  * @return {Promise.<{Array.<HistoryObject>}>}
  */
getWithdrawHistory(currency, id = null) {}
```

- 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 "completed")
 * @property {string} address - destination address
 * @property {string} hash - transaction hash
 */

/**
  * Get deposit history
  * @param {string} currency - currency, example: "BTC", "ETH"
  * @param id - if specified, method returns data of specific withdraw
transaction
  * @return {Promise.<{Array<HistoryObject>}>}
  */
getDepositHistory(currency, id = null) {}
```

- getBalance - получение баланса

```
/**
  * Get exchange balance (indexes are currencies, values are amounts)
  * @param {Object} ws - WebSocket object
  * @return {Promise.<{Array<string, Number>}>}
  */
getBalance(ws) {}
```

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

```
/**
* Make internal transfer between internal accounts
* @abstract
* @param {string} currency - cryptocurrency
* @param amount - amount of transfer
* @param type - type of transfer ("exchangeToMain", "mainToExchange",
...)
* @return {Promise.<TransactionObject>}
*/
makeInternalTransfer(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.

- getTradeHistory - получение данных по выполненным ордерам

```
/**
  * @typedef {Object} TradeObject
  * @property {number} id - trade id
  * @property {number} orderId - order id
  * @property {number} symbol - exchange symbol
  * @property {number} price - price
  * @property {amount} amount - amount
  * @property {number} timestamp - trade timestamp
  * @property {string} type - 'buy' or 'sell'
  * @property {number} fee - trade fee
  */

/**
  * Get all trades of specific order ID and symbol
  * @param {WSObject} wsData - WebSocket connection data
  * @param {number|string|null} orderId - order ID (if not null,
filter by order ID)
  * @param {null|string} symbol - exchange symbol (if not null, filter
by symbol)
  * @return {Promise.<{Array<TradeObject>}>}
  */
getTradeHistory(wsData, orderId = null, symbol = null) {}
```

Метод getTradeHistory возвращает массив трейдов, отфильтрованный по order ID или по символу.

Если один из вышеперечисленных методов нельзя определить как описано выше, тогда нужно об этом сообщить для уточнения анотации 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 на примере биржи Bitfinex (https://www.bitfinex.com/).

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