

**Михаил Шардин**

личный блог



27 ноября 2024, 04:35

[+ Подписаться](#)

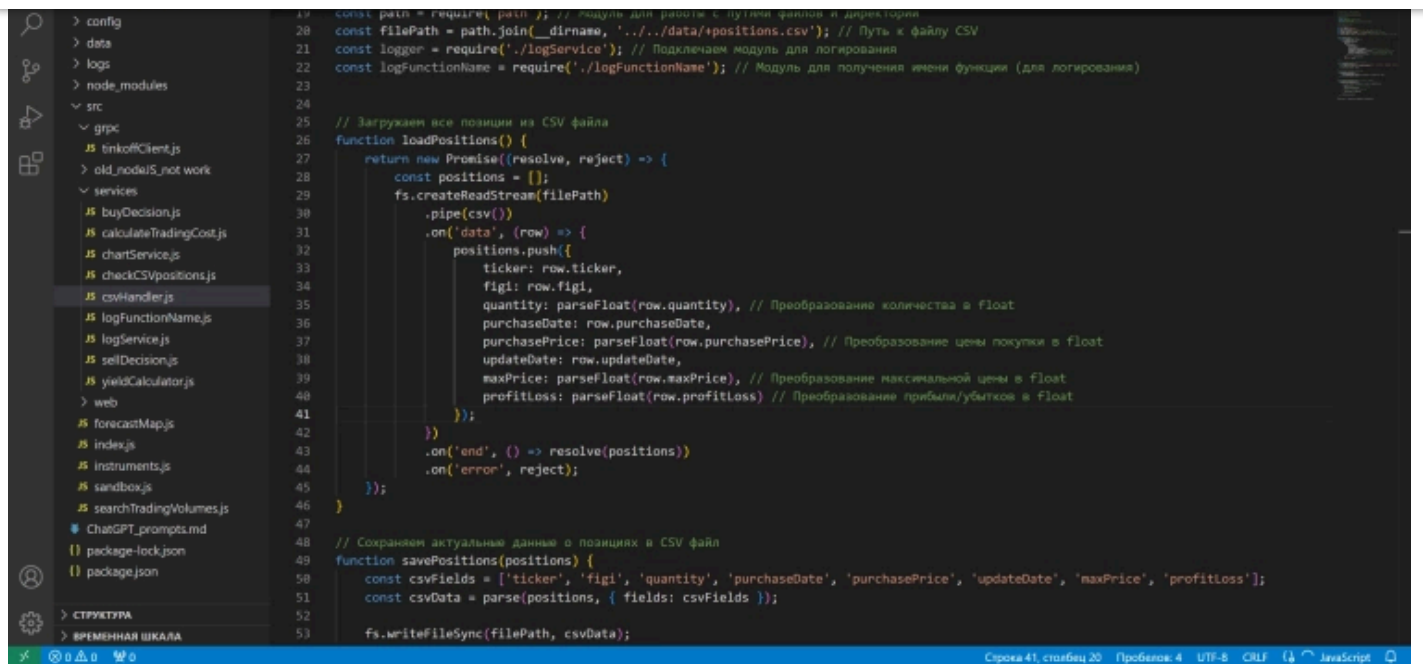
## Отслеживание позиций торгового робота Московской биржи через CSV файл

Нахожусь в процессе написания механизма торгового робота, работающего на Московской бирже через API одного из брокеров. Брокеров имеющих своё АПИ для МосБиржи катастрофически мало — мне известно только о трёх. При этом, когда я стал публиковать модули робота (и полностью [выложу готовый механизм робота на GitHub](#)), то стал получать непонимание — например, мне писали в комментариях — зачем придумывать велосипед, когда уже есть QUIK — популярная российская платформа для биржевых торгов. В Квике уже есть готовый функционал «импорт транзакций из файла» или таблица «карман транзакций». В тех же комментариях предлагали даже рассмотреть использование платформы 1С для робота, но оказалось, что торговля все равно будет осуществляться через импорт .tri-файла в Квик.

Лично мне Квик не очень нравится тем, что это программа для Windows. Хочется иметь механизм торгового робота, который был бы кроссплатформенным и легким — это позволит использовать его даже на «слабом» сервере. К тому же, [много лет назад, когда Квик был единственной альтернативой для частного лица](#), невозможно было внутри одной Windows без использования виртуальной машины запустить несколько копий программы технического анализа с разными системами — для того, чтобы каждая из этих копий отправляла свои сигналы на покупку и продажу в соответствующий Квик. Это было нужно для разных торговых стратегий.

По субъективным причинам я стал писать [торгового робота в среде исполнения JavaScript Node.js](#), но для тестирования на истории пришлось [использовать Python и его библиотеки](#).

Введите текст комментария



```
17 const path = require('path'); // Модуль для работы с путями файлов и директориями
18 const filePath = path.join(__dirname, '../data/positions.csv'); // Путь к файлу CSV
19 const logger = require('./logService'); // Подключаем модуль для логирования
20 const logFunctionName = require('./logFunctionName'); // Модуль для получения имени функции (для логирования)
21
22 // Загружаем все позиции из CSV файла
23 function loadPositions() {
24   return new Promise((resolve, reject) => {
25     const positions = [];
26     fs.createReadStream(filePath)
27       .pipe(csv())
28       .on('data', (row) => {
29         positions.push({
30           ticker: row.ticker,
31           figi: row.figi,
32           quantity: parseFloat(row.quantity), // Преобразование количества в float
33           purchaseDate: row.purchaseDate,
34           purchasePrice: parseFloat(row.purchasePrice), // Преобразование цены покупки в float
35           updateDate: row.updateDate,
36           maxPrice: parseFloat(row.maxPrice), // Преобразование максимальной цены в float
37           profitLoss: parseFloat(row.profitLoss) // Преобразование прибыли/убытков в float
38         });
39       })
40       .on('end', () => resolve(positions))
41       .on('error', reject);
42   });
43 }
44
45 // Сохраняем актуальные данные о позициях в CSV файл
46 function savePositions(positions) {
47   const csvFields = ['ticker', 'figi', 'quantity', 'purchaseDate', 'purchasePrice', 'updateDate', 'maxPrice', 'profitLoss'];
48   const csvData = parse(positions, { fields: csvFields });
49   fs.writeFileSync(filePath, csvData);
50 }
```

## Проблемы с записью позиций в Node.js

Вообще именно этот модуль пришлось пару раз переписывать, потому что не смог сразу отладить его. Проблема была в том, что вызов модуля записи и обновления позиций осуществлялся сразу из нескольких мест и одни результаты перезаписывали другие. Но удалось разобраться и теперь всё протестировано и работает.

Дополнительно использую библиотеки `csv-parser` и `json2csv` — это популярные инструменты Node.js для обработки данных CSV, каждая из которых служит различным целям:

- `csv-parser`, это легкая и быстрая библиотека для анализа файлов CSV. Она основана на потоках, что делает ее очень эффективной для обработки больших наборов данных.
- `json2csv`, это утилита для преобразования данных JSON в формат CSV. Идеально подходит для экспорта данных из приложений в структуру, удобную для CSV, может работать как синхронно, так и асинхронно.

Установка этих библиотек:



```
npm install csv-parser json2csv
```

## Мой модуль csvHandler.js

Этот код определяет модуль для взаимодействия с CSV-файлом для управления финансовыми торговыми позициями. Служит для загрузки, сохранения, обновления и удаления финансовых позиций, хранящихся в CSV-файле.

### Ключевые библиотеки:

- fs: для операций файловой системы, таких как чтение и запись файлов.
- csv-parser: для анализа CSV-файлов в объекты JavaScript.
- json2csv: для преобразования объектов JavaScript в формат CSV для сохранения.
- path: для управления путями к файлам.
- Интеграция: включает пользовательские модули для ведения журнала (logService) и получения имен функций для лучшей отладки.

### Функциональность

1. Обработка пути к файлу: использует модуль path для поиска CSV-файла, хранящего данные о позиции: ../../data/+positions.csv.

1. Функции управления позицией:

#### loadPositions():

1. Считывает CSV-файл и анализирует его в массив объектов позиции.
2. Преобразует числовые поля (quantity, purchasePrice, maxPrice, profitLoss) в числа с плавающей точкой для вычислений.
3. Возвращает обещание, которое разрешается с проанализированными данными или отклоняется в случае ошибки.

#### savePositions(positions):

---

`removePosition(figi):`

---

1. Удаляет позицию из CSV-файла на основе ее figi (уникального идентификатора).
2. Загружает все позиции, отфильтровывает указанную и перезаписывает файл.

`updatePosition(newPosition):`

1. Добавляет новую позицию или обновляет существующую в CSV-файле:
  2. Если figi существует, обновляет соответствующую позицию.
  3. В противном случае добавляет новую позицию.
  4. Сохраняет обновленный список обратно в CSV-файл.
1. Экспортированные модули: функции `loadPositions`, `updatePosition` и `removePosition` для использования в других частях робота.

Полный код `csvHandler.js`:

```

const fs = require('fs');
const csv = require('csv-parser');
const { parse } = require('json2csv');
const path = require('path'); // Модуль для работы с путями файлов и директорий
const filePath = path.join(__dirname, '../data/positions.csv'); // Путь к файлу CSV
const logger = require('./logService'); // Подключаем модуль для логирования
const logFunctionName = require('./logFunctionName'); // Модуль для получения имени функции (для логирования)

// Загружаем все позиции из CSV файла
function loadPositions() {
  return new Promise((resolve, reject) => {
    const positions = [];
    fs.createReadStream(filePath)
      .pipe(csv())
      .on('data', (row) => {
        positions.push({
          ticker: row.ticker,
          figi: row.figi,
          quantity: parseFloat(row.quantity), // Преобразование количества в float
          purchaseDate: row.purchaseDate,
          purchasePrice: parseFloat(row.purchasePrice), // Преобразование цены покупки в float
          updateDate: row.updateDate,
          maxPrice: parseFloat(row.maxPrice), // Преобразование максимальной цены в float
          profitLoss: parseFloat(row.profitLoss) // Преобразование прибыли/убытков в float
        });
      })
      .on('end', () => resolve(positions))
      .on('error', reject);
  });
}

// Сохраняем актуальные данные о позициях в CSV файл
function savePositions(positions) {
  const csvFields = ['ticker', 'figi', 'quantity', 'purchaseDate', 'purchasePrice', 'updateDate', 'maxPrice', 'profitLoss'];
  const csvData = parse(positions, { fields: csvFields });

  fs.writeFileSync(filePath, csvData);
}

// Удаляем позицию из CSV файла (после продажи)
function removePosition(figi) {
  loadPositions().then(positions => {
    const updatedPositions = positions.filter(position => position.figi !== figi);
    savePositions(updatedPositions);
  });
}

// Добавляем новую позицию или обновляем существующую в CSV файле
function updatePosition(newPosition) {
  loadPositions().then(positions => {
    const index = positions.findIndex(pos => pos.figi === newPosition.figi);

    if (index === -1) {
      // Добавляем, если не нашли существующую позицию
      positions.push(newPosition);
    } else {
      // Обновляем, если позиция уже существует
      positions[index] = newPosition;
    }

    savePositions(positions);
  });
}

```

## Мой модуль `checkCSVpositions.js`

Этот модуль важен для обеспечения согласованности данных между локальным CSV-файлом и текущими позициями, полученными из T-Bank Invest API. Он проверяет наличие несоответствий, которые могут привести к ошибкам в торговых операциях, и останавливает робота, если обнаруживаются несоответствия.

### Основные функции

#### 1. Интеграция с внешними системами

- T-Bank Invest API: взаимодействует с API для извлечения торговых позиций в реальном времени.
- CSV File Management: использует локальный CSV-файл для хранения и управления представлением бота о торговых позициях.

#### 1. Проверка согласованности

- Сравнивает позиции из CSV-файла с позициями с сервера T-Bank Invest API.
- Проверяет как количество, так и наличие позиций для обнаружения несоответствий.

#### 1. Обработка ошибок

- Регистрирует подробные ошибки при обнаружении несоответствий.
- Останавливает торговые операции для предотвращения дальнейших действий на основе неверных данных.

### Основные функции:

## 1. `getServerPositions()`

- Извлекает все открытые позиции из T-Bank Invest API.
- Извлекает позиции с ценными бумагами и преобразует баланс в float для сравнения.
- Регистрирует ответ сервера для отладки и аудита.

## 2. `checkForDiscrepancies()`

- Загружает данные CSV: считывает локальную запись позиций бота с помощью `csvHandler`. Сравнивает позиции:
- Для каждой позиции CSV ищет соответствующую позицию на сервере с помощью FIGI (уникальный идентификатор).
- Извлекает размер лота для точного сравнения количества.
- Если обнаружены расхождения в количестве или отсутствующие позиции, регистрирует ошибки и останавливает торговлю.
- Статус журнала: подтверждает, когда все позиции совпадают, и позволяет продолжить торговлю.

### Рабочий процесс

#### 1. Извлечение позиций:

- Локальные позиции загружаются из CSV-файла

---

3. Сравнивает с балансом на сервере. Ошибки регистрируются, если:

4. Количества не совпадают.

5. Позиция в CSV-файле отсутствует на сервере.

1. Безопасность робота:

- Любые обнаруженные расхождения вызывают ошибку, останавливающую торговые операции.
- Не позволяет роботу совершать сделки на основе устаревших или неверных данных.

Полный код `checkCSVpositions.js`:



```

const logger = require('./logService'); // Логирование в файл и консоль
const logFunctionName = require('./logFunctionName'); // Получение имени функции

const secrets = require('./../config/secrets'); // Ключи доступа и идентификаторы
const config = require('./../config/config'); // Параметры
const csvHandler = require('./csvHandler'); // Работа с CSV файлами

const TinkoffClient = require('./../grpc/tinkoffClient'); // Модуль для взаимодействия с API Tinkoff
Invest
const API_TOKEN = secrets.TbankSandboxMode;
const tinkoffClient = new TinkoffClient(API_TOKEN);

// Функция для получения всех позиций с сервера
async function getServerPositions() {
  try {
    const accountId = {
      accountId: secrets.AccountID
    };
    const response = await tinkoffClient.callApi('OperationsService/GetPositions', accountId);

    // Логруем полученные позиции с сервера
    logger.info(`Все открытые позиции счета ${secrets.AccountID}: \n ${JSON.stringify(response,
null, '\t')}\n\n`);

    // Возвращаем только позиции с ценными бумагами (securities)
    return response.securities.map(sec => ({
      figi: sec.figi,
      balance: parseFloat(sec.balance) // Преобразуем баланс в float
    }));
  } catch (error) {
    logger.error(`Ошибка при получении позиций с сервера: ${error.message}`);
    throw error;
  }
}

// Функция для проверки расхождений
async function checkForDiscrepancies() {
  try {
    // Загружаем текущие позиции из CSV файла
    var csvPositions = await csvHandler.loadPositions();

    // Получаем позиции с сервера
    const serverPositions = await getServerPositions();

    // Проверяем каждую позицию из CSV
    for (const csvPosition of csvPositions) {
      // Находим соответствующую позицию с сервера
      const serverPosition = serverPositions.find(pos => pos.figi === csvPosition.figi);

      if (serverPosition) {
        const lotSize = await tinkoffClient.getLot(csvPosition.figi);
        logger.info(`Количество бумаг в лоте ${csvPosition.figi}: ${lotSize} шт.`);
        const csvTotal = csvPosition.quantity * lotSize;

        // Сравниваем количество позиций
        if (csvTotal !== serverPosition.balance) {
          // Если есть расхождение, логируем ошибку и останавливаем торгового робота
          logger.error(`Ошибка: Несоответствие по FIGI ${csvPosition.figi}. CSV: ${csvTotal},
Сервер: ${serverPosition.balance}`);
          throw new Error('Найдено несоответствие позиций. Остановка торговли.');
```



```
// Экспортируем функции
module.exports = {
  checkForDiscrepancies
};

// checkForDiscrepancies().catch(logger.error);
```

## Итоги

Проект полностью представлен на Гитхабе: <https://github.com/empenoso/SilverFir-TradingBot>. Новые модули будут загружаться по мере написания и тестирования.

Автор: [Михаил Шардин](#)

27 ноября 2024 г.

торговые роботы

6K

☆ 11    💬 12    ❤️ 18



**Михаил Шар...**

📍 Пермь

👤 41    📊 467

📅 с 23 января 2019

✉️ [empenoso](#)

+ Подписаться

12 КОММЕНТАРИЕВ

Сначала старые ▾



**T-800**

27 ноября 2024, 06:50

В Квике можно не только через текстовые \*.tri файлы работать, но и через Trans2Quik.dll  
А вообще, удачи в вашем начинании.

↩️ 👍 +1 💬



**akumidv**

27 ноября 2024, 07:39

По мере усложнения логики код будет сложно поддерживать. Надо разделять на независимые модули.

Сохранение не потокобезопасное — будут ошибки при синхронной записи и считать. Очередь поможет.

И новые функции лучше разрабатывать через тесты — тестирование позволит делать менее связанный код, его потом легче поддерживать и модифицировать. Ну а тесты позволят прогонять функционал и быть более уверенным, что работает.

Из простых решений еще можно ESLint добавить в проект — он будет помогать ошибки находить и стиль поддерживать.

Сам JS для бота сложный выбор. Проще питон подучить и на нем делать. Там хотя бы pandas и numba с Jit копиляцией есть и мультипроцессорность понятная.

PS логи и данные в проекте — зло.



Iliam

27 ноября 2024, 07:43



QUIK конечно поразительная программа, в эпоху санкций, работающая на Windows. За 20 лет, могли бы и кроссплатформенность добавить. А вам удачи.



SergeyJu

27 ноября 2024, 10:07



У квика всегда были конкуренты, которые так и не распространились.

У гуты-банка, у Цериха, у Атона, у Альфы, у Айти-инвеста, был еще нет-трейдер и это неполный список.

И, насколько я знаю, из под линукса на квике торгуют.

Что касается авторского проекта, если рассматривать его как рабочий макет, то все здорово. Осталось понять косяки, учесть принципиальные недостатки и написать заново.

Автор, как я понимаю, пишет быстро, так что все в его руках.



Михаил Шардин

27 ноября 2024, 13:03



SergeyJu, линукса на квике скорее всего через эмулятор. Тот же Wine например



SergeyJu



счета разложить. Например, на одном создать пассивный портфель. А торговать на втором активно.

Да, и учтите, данные об транзакциях могут иногда теряться. Систем должна быть устойчива к максимально большому спектру искажений в данных. Не в смысле частоты появления, а в смысле критичности для портфеля.

Быстрый робот моего знакомого как-то, теряя подтверждения о сделках, считал, что сделка не проведена и долбал снова и снова, пока не вышел на максимально возможное плечо.

**Denis**

27 ноября 2024, 14:34



Михаил Шардин, Wine не эмулятор, а имплементация подмножества Win API для запуска под X. Для маков тоже должен быть.

Гораздо проще использовать нативный Офис/Excel, да и Квик от них ничем не отличается. Просто рисуется все не в винде, а в Линуксе.

**Константин Лебедев** ★

27 ноября 2024, 11:24



Тебя T-bank дурит и на самом деле об заявки обрабатывает у себя для выхода на прямую лучше использовать [github.com/kmlebedev/txmlconnector](https://github.com/kmlebedev/txmlconnector)

Да он немного легаси, но это прямо прямой доступ, там есть пример как данные выгрузить в клик, а дальше можно с ними вертеть как захочешь.

**Slan**

27 ноября 2024, 13:13



что мешает в json формате хранить данные? их и получать обратно проще нативными методами. зачем эта компиляция в csv туда-сюда? Ну и раз предполагается хранение — не имеет ли смысл использовать базу данных?

**SergeyJu**

27 ноября 2024, 13:17



Slan, конечно, данные должны быть в базе. И структура базы не самое последнее дело при проектировании.



**Beach Bunny**

27 ноября 2024, 21:37



Для Backtrader есть готовы коннектор для Тинька, смысл писать велосипед на JS  
[github.com/cia76/TinkoffPy](https://github.com/cia76/TinkoffPy)



Напишите комментарий...



ОТПРАВИТЬ

Установите приложение Смартлаба:



Google Play



App Store



AppGallery



RuStore

[О смартлабе](#)[Реклама](#)[Полная версия](#)

